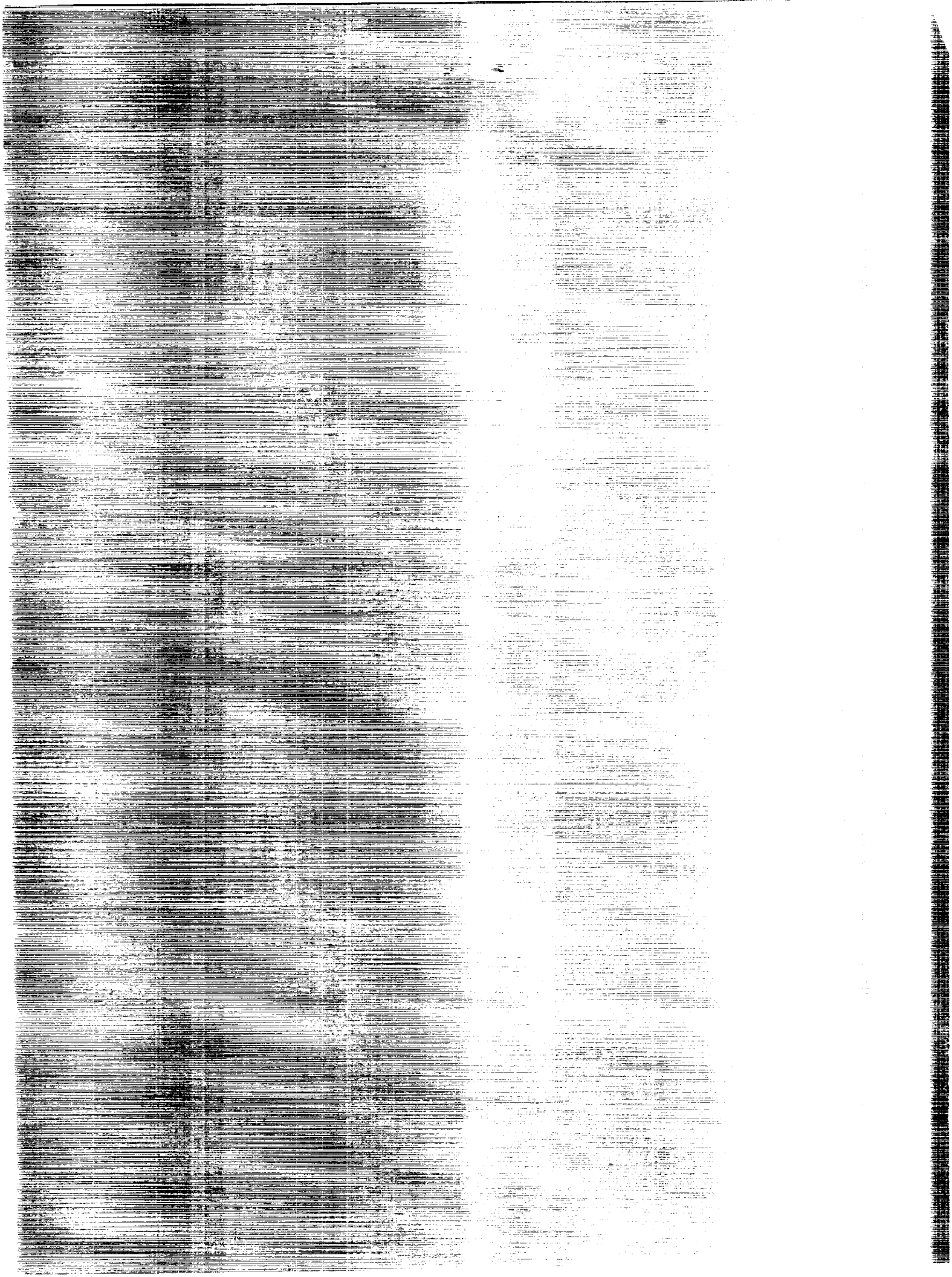
NASA Contractor Repo

Rule Groupings: ftware Engineering Appr h Towards Verification of Ex ystems

Mala Mehrotra

CONTRACT NAS1-18585
MAY 1991



NASA Contractor Report 4372

Rule Groupings: A Software Engineering Approach Towards Verification of Expert Systems

Mala Mehrotra
ViGYAN, Inc.
Hampton, Virginia

Prepared for
Langley Research Center
under Contract NAS1-18585

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division

1991

Rule Groupings: A Software Engineering Approach towards Verification of Expert Systems*

Mala Mehrotra

Vigyan Inc.

Hampton, Va 23665.

Abstract

Currently, most expert system shells do not address software engineering issues for developing or maintaining expert systems. As a result, large expert systems tend to be incomprehensible, difficult to debug or modify and almost impossible to verify or validate. Partitioning rule-based systems into rule groups which reflect the underlying subdomains of the problem should enhance the comprehensibility, maintainability and reliability of expert-system software. In this paper, we elaborate our attempts to semi-automatically structure a CLIPS rule base into groups of related rules that carry the same type of information. Different distance metrics that capture relevant information from the rules for grouping are discussed. Two clustering algorithms that partition the rule base into groups of related rules are given. Two independent evaluation criteria are developed to measure the effectiveness of the grouping strategies. Results of our experiment with three sample rule bases are presented.

*This research was supported by NASA Contract No. NAS1-18585 at NASA Langley Research Center, Hampton, VA 23665.

1 Introduction

Knowledge-based expert systems are playing an increasingly important role in NASA space and aircraft systems. They have potential usefulness in fault diagnostics and recovery, monitoring, planning, scheduling, and control systems, and are already being used as ground-based advisory systems. However, many of NASA's software applications are life- or mission-critical, and knowledge-based systems do not lend themselves to the traditional verification and validation techniques for highly reliable software[1].

Even relatively small rule-based systems contain hundreds of rules, and as the number of rules increases, the number of possible interactions between rules increases exponentially. There is also another dimension of complexity independent of the number of rules in the system. This is the number of potential matches for each pattern in a rule. As the complexity of these patterns increases, the number of possible combinations of facts required for testing these patterns increases exponentially. Therefore, it is infeasible to attempt exhaustive testing of every possible path through the system, much less every possible combination of inputs. Thus, exhaustive testing is impractical for demonstrating high reliability of knowledge-based systems, and less computationally intensive analysis techniques must be employed.

Although the meaning of each rule in isolation may be well understood, the complexity of the rule-based system stems from the interactions and interdependencies between rules. Therefore, analysis of the system should focus on understanding those interactions and assuring that they are correct. The approach we have chosen is to investigate the structuring of a rule-based system into a set of groups. Such groups would allow one to abstract away from the *each rule is a procedure call* point of view, and look at the system from a higher semantic level. This should make explicit the underlying subdomains of the problem and also aid in understanding the level of knowledge (i.e. deep or shallow reasoning) that was applied to solve the subproblems. The groupings are made to aid in comprehension of the

rule base and have no effect on the execution of the system.

In this paper we describe our attempts to extract the domain knowledge from a rule base by structuring the rule base into groups of related rules. We present several distance metrics which measure the relatedness of rules, and we describe two different clustering algorithms.

Related Work

The issues involved in grouping rules have been studied at various levels. Lindell[2] suggests a clustering algorithm based on keywords contained in comments. Clearly this can work only for well-documented rule bases and for those where one can succeed in finding appropriate keywords.

Lindenmayer, et al.[3] have proposed a certain methodology for grouping OPS5 rule bases for the Hubble Space Telescope Project. Different rule couplings are discussed with respect to *make*, *remove* and *modify* constructs in OPS5. However, dependency between two rules is assumed to exist only from the consequent of one rule to the antecedent of another.

A more sophisticated approach, by Jacob and Froscher[4, 5], utilizes more forms of data dependencies of a rule base and groups rules on their informational content. However, Jacob's grouping strategy also assumes that domain knowledge is in the form of a decision tree, in which chaining of rules takes place predominantly from the consequent of one rule to the antecedent of another. It presumes a hierarchical decomposition of domain knowledge, which seems to be well suited for classification problems, such as the animal identification rule base. However, it does not work well in monitoring or scheduling problems, where the antecedents of rules generally carry domain information, such as different modes of failure, while consequents usually give only directives for taking the appropriate actions.

An Overview of Our Approach

We have taken a pattern-matching approach towards grouping of rules[6, 7]. In this approach, the commonality of items in the rules determines the distance between them. Our rule grouping process consists of three stages. First, the distance between each pair of rules is computed and stored in a distance matrix. In the second stage, the computed distances are modified so that all distances satisfy the triangle inequality. That is, we replace the distance between two rules by a shorter distance, if there exists an intermediate rule through which a shorter path can be created. Consider three rules r_i, r_j and r_k with inter-rule distances d_{ij}, d_{ik} and d_{jk}. If $d_{ij} + d_{jk} < d_{ik}$, then we replace d_{ik} by $d_{ij} + d_{jk}$. This method thus extracts the transitive dependency between rules. Finally, we apply a clustering algorithm to form our groups. There are two approaches we consider for clustering. The first one is an automatic clustering algorithm based on graph-partitioning approaches. The second requires the user to designate certain rules as "primary rules" or "seed rules" around which the clustering algorithm will form groups.

Automatic clustering algorithms work, but optimal clustering is NP-complete, and the resulting clusters may not conform, in any reasonable way, to the clusters a user would desire. Thus, we developed a more user-directed approach, in which the user designates some rules as primary rules, and a cluster is automatically formed around each of these rules. These rules typically reflect key concepts from the domain; thus, the resulting clusters correspond closely to what the user desires.

The rest of the paper is structured as follows. In the second section, we discuss the design issues in developing different distance metrics. In section 3, the two different algorithms for clustering are given. Section 4 describes our experiments on three different rule bases. This section also discusses various quality measures for determining the "goodness" of groupings. Based on these, a performance evaluation is done of the different distance metrics on three sample rule bases. Section 5 discusses the relationship of our efforts to verification and validation issues. Finally, we present our conclusions and

future directions for research.

2 Design of Distance Metric

In order to allow our system to adapt to different types of rule bases, we have designed different distance metrics * for measuring the relatedness of rules. Different metrics capture different kinds of information from the rule base. In this section, we present the motivation for our different distance metrics and the criteria used in designing them.

Experience with using Jacob and Froscher's grouping algorithm on rule-based systems, such as ONAV[†],and MMU-FDIR[‡], suggests that different application domains require different distance metrics. Finding a single universal metric appears to be impossible, since different expert systems require one to capture different information.

In our discussions henceforth, we follow the terminology of Giarratano and Riley[10]. A rule base is made up of rules with antecedents and consequents. Each antecedent or consequent is composed of patterns which match facts in working memory at run time. Each pattern is further divided into fields or tokens. In CLIPS [§] [11] a token can be a word, string or number. We refer to these as *items.*

Only certain items are relevant for grouping. We ignore all run-time aspects of the rule base, since the presence of a particular item in a rule appears to be more important to proper grouping than the way in which it is used. Also, analysis of the run-time behavior of a rule base could be prohibitively expensive, amounting to direct simulation of all logic paths. Our grouping does not functionally alter the rule base at all; once the grouping is performed, the user sees rules in their original form. In effect, we are trying to automate a user's first steps in attempting to comprehend an unknown rule base.

In order to do effective grouping, we need to suppress all information conveyed by key words, such

*The usage of term *metrics* in this paper does not conform to the strict mathematical definition of metrics.
†Onboard Navigation Expert System [8]
‡Manned Maneuvering Unit - Fault Detection, Isolation and Recovery System [9]
§'C' Language Production System

as, *retract, bind, test*; they serve only as run-time directives. For example, a rule asserting a fact A, and another one testing for the absence of A, would not be temporally grouped together. But statically, since both rules are referring to the same information, their meaning may be more clear if they are present in the same group. If the fact A gets modified, then it is important to show that the validity of the rule testing for the absence of A still holds. A similar justification holds for ignoring keywords. We do, however, include numeric- and word-type fields in our metrics, since they frequently contain domain-specific information relevant to static grouping. However, we suppress all strings and variable names. Strings rarely occur outside of print statements and convey little domain information. Variables are local to each rule and hence cannot carry information pertinent to the static relationship of rules. Their bindings take place at run time, and therefore no values can be assigned to them *a priori*.

Given these fundamental design choices, we can now state a generic definition of the distance between two rules, r_1 and r_2, as

$$d(r_1, r_2) = \frac{Total\ no.\ of\ items\ in\ r_1\ and\ r_2}{no.\ of\ \text{“}common\text{”}\ items\ in\ r_1\ and\ r_2}$$

where different definitions of "common" give rise to different distance metrics. When there are no common items between r_1 and r_2, $d(r_1, r_2)$ is replaced by the maximum number of patterns allowed.

The nature of the domain knowledge enforces a certain programming methodology on the developer of a rule base. Classification systems, such as the animal identification problem [12], have a hierarchical structure which yields easily to a data-flow grouping like Jacob's and Lindenmeyer's. Classification of disease hierarchies also falls in the same application type. The fundamental characteristic of such systems is that the flow of data takes place from the consequent of one rule to antecedents of other rules. Three rules from the animal identification rule base are presented in Figure 1 to show the data-flow aspects of the system. For this type of rule base, it is appropriate to use a distance metric, $d_{df}(r_1, r_2)$,

```
( defrule r1
        (animal gives milk)
  =>
        ( assert(animal is-a  mammal))   )

( defrule r2
        (animal is-a  mammal)
        (animal has hoofs)
  =>
        ( assert(animal is-a  ungulate))  )

( defrule r3
        (animal is-a  ungulate)
        (animal has blackstripes)
  =>
        ( assert(animal is-a  zebra))  )
```

Figure 1: Example Rules from Animal Identification Rule Base

between two rules, r_1 and r_2, defined as,

$$d_{df}(r_1, r_2) = \frac{Total\ no.\ of\ items\ in\ consequent\ of\ r_1\ and\ antecedent\ of\ r_2}{no.\ of\ \text{"common"}\ items\ in\ consequent\ of\ r_1\ and\ antecedent\ of\ r_2}.$$

A monitoring system issues different commands depending on the status of different components of the system being monitored. In such systems, the antecedents of the rules usually search for special values of flags in the component system, and the consequents assert actions to be taken when different components fail. Example rules from MMU-FDIR given in Figure 2 illustrate this point.

The bulk of domain information required for grouping is usually present in the antecedents of rules in a monitoring system. This gives rise to the antecedent distance metric:

$$d_{ant}(r_1, r_2) = \frac{Total\ no.\ of\ items\ in\ antecedents\ of\ r_1\ and\ r_2}{no.\ of\ \text{"common"}\ items\ in\ antecedents\ of\ r_1\ and\ of\ r_2}.$$

If one wanted to group on different component failures asserted by the consequents of such rules,

```
( defrule cea-a-gyro-input-pitch-pos-2
        (aah on)  (gyro on)
        (gyro movement pitch pos)
        (side a on) (side b on)
        (rhc roll none pitch none yaw none)
        (thc x none y none z none)
        (vda a ?m on)
  =>
        ( assert (failure cea))
        ( assert (suspect a))  )
```

Figure 2: Example Rule from MMU-FDIR Rule Base

a consequent distance metric could be defined as well. The antecedents in the above distance metric would be replaced by the consequents.

In a diagnostic system, a data-flow aspect is present together with a monitoring aspect. A hierarchical search space reflects the different sub-domains of the problem. Rule interdependency is from the consequent of one rule to the antecedent of another. Certain other rules are related to each other through antecedents alone, since they detect similar symptoms from the domain. An example is given in Figure 3 where the antecedents of the first and second rules need to be fired by a situation match from working memory. Other rules in the rule base do not assert these patterns. However, there is data-flow dependency between the consequent of the first rule and the antecedent of the third rule. Since such is the case for most diagnostic systems, both sides of the rule deserve consideration. Therefore, we have the following metric to define distance in such a case:

$$d_{all}(r_1, r_2) = \frac{Total\ no.\ of\ items\ in\ r_1\ and\ r_2}{no.\ of\ \text{“}common\text{”}\ items\ in\ r_1\ and\ r_2}.$$

```
( defrule starter_ok
        (starter cranks_engine yes)
        (lights dim slightly)
=>
        ( assert(battery problems no))
        ( assert(engine is_tight no))    )

( defrule battery_not_ok
        (starter cranks_engine yes)
        (lights dim considerably)
=>
        ( assert(battery problems yes))
        ( assert(engine is_tight yes))   )

( defrule engine_misfires
        (battery problems no)
        (engine is_tight no)
        (engine misfires constantly)
=>
        ( assert(cylinder problems yes))  )
```

Figure 3: Example Rules from Car Diagnostics Rule Base

According to the distance metrics defined above, relatedness of two rules is inversely proportional to the distance between them. That is, the more related two rules are, the less the distance between them.

3 Clustering Algorithm

In order to be useful, a clustering algorithm must break up a rule base into meaningful groups. However, which particular clustering will prove "meaningful" is strongly dependent on the nature of the rule base. To address this issue, we perform clustering based on the different distance metrics. By carefully choosing the right metric, we hope to be able to achieve a clustering well suited to each particular rule base.

3.1 Automatic Clustering Algorithm

The automatic clustering algorithm starts with each rule in its own group. Groups are then merged based on the minimum inter-group distance. Here, we define inter-group distance, $D(i,j)$, as follows:

$$D(i,j) = \frac{\sum_{r_k \epsilon\ G_i} \sum_{r_l \epsilon\ G_j} d(r_k, r_l)}{n_i * n_j}$$

where n_i and n_j are the number of rules in groups G_i and G_j, respectively.

Using this definition of inter-group distance, we form an automatic clustering algorithm as follows. In this algorithm, the user provides the total number of groups, M, to be formed, which serves as a stopping criterion. A high-level view of the algorithm is given below:

Initialize each rule into its own group

While ($number\ of\ groups\ >\ M$)

  Find groups G_i and G_j with minimum inter-group distance $D(i,j)$

  Merge groups G_i and G_j

We experimented with various other stopping criteria which did not need any user input. However, none of them seem to work consistently across rule bases. For example, we calculated the mean distance between the rules in the rule base, and when the intergroup distance exceeded the mean distance, the algorithm stopped clustering. A similar experiment was done taking the standard deviation of rules as the stopping criterion. In some instances, the rule base would fragment into very small meaningless groups; at other times, one would obtain a very skewed grouping with some very large groups and some very small ones. Since the purpose of our study is to ascertain which distance metric gives the right grouping for a rule base, penalizing the distance metric for not responding well to a stopping criteria

did not seem justified. Hence, we chose the option of requiring the user to input the number of groups desired and then seeing which distance metric fares best.

As we shall discuss later, it is very difficult to define an optimal grouping for a rule base. Even if one could be defined, implementation of a guaranteed optimal grouping solution for the rule base would reduce to an optimal graph partitioning problem, which is known to be NP-complete.

3.2 Clustering through Primary Rules Selection

In this strategy, more input by the user is required. Not only is the number of groups provided by the user, but for each group, a primary rule must be given which captures a concept from the rule base. These rules then form the seed or context around which clustering can take place.

A high-level view of the clustering algorithm through primary rule selection is as follows:

For each rule r_i

Find primary rule r_p for which $d(r_i, r_p)$ is minimized

Merge rule r_i into $G(r_p)$

In this algorithm, rules that are equidistant from two or more primary rules are skipped over in the first pass when all the other rules are clustered. In the second pass, the equidistant rules are resolved based on the criterion that they minimize the average distance between rules in the group that they are pulled into.

4 Experimental Results

In order to assess these different approaches to clustering, we tested our algorithms on three sample rule bases. The largest one is the MMU-FDIR, written by McDonnell Douglas Astronautics, having 104 rules. Correct thruster configurations for the two sides of the Manned Maneuvering Unit and gyroscope in the primary and backup modes are checked by 73 rules. An additional 14 rules deal with

failure recovery for the two control electronics assemblies, and 7 rules deal with tankthruster tests. The remaining 10 rules do printing and demonstration. These 10 rules were removed before we began grouping, for reasons that have been mentioned in Section 2. The majority of the rules in this system have a similar structure. Each antecedent consists of tests for automatic attitude hold, gyroscope readings, rotational and translational hand controller values, and valve drive amplifier outputs. The consequent then declares the faulty subsystems.

The second rule base is a car diagnostic expert system we wrote. It has 60 rules to diagnose problems in 12 subsystems of the car. Typical subsystems include the distributor, carburetor, and ignition systems. This rule base traverses a search tree which branches first at the root, based on the status of the headlights and whether or not the starter can crank the engine. The search is guided by various observations of ammeter readings, lights, spark-plug reversals and others. One of the problems in this rule base is that rules assert several potential problems. This feature greatly complicates grouping, but seems typical of fault diagnostic systems. Such complicated couplings mirror the complexity of the underlying domain.

Our last rule base is the animal classification program[12], a well-known problem in artificial intelligence. It contains 14 rules, which classify an unknown animal as either an ungulate, carnivore, or bird. Then, having placed a given animal into one of these broad categories, it makes a more precise classification, deciding, for example, whether an animal is a cheetah or a tiger, based on detailed characteristics.

A listing of the above rule bases and the groupings obtained are detailed in the Appendices.

4.1 Evaluation Criteria

It is difficult to judge the quality of a rule base grouping; judgement of quality is, in general, highly subjective. Nonetheless, it appears essential to assess the value of our grouping strategies in a com-

paratively rigorous manner. Otherwise, we would be unable to make definitive statements about the comparative merits of these strategies and of the value of grouping, more generally.

In order to measure the quality of groupings, we have developed two independent measures. The first of these measures is based on an "ideal grouping," which we generate ahead of time by hand. It measures the deviation of the computed grouping from our ideal grouping. The second of these measures the "stability" of groupings obtained by the primary rule algorithm, where we measure stability by counting the number of rules which migrate between groups as we vary our choice of primary rules.

While some attempt at measuring the quality of a grouping is necessary, there are difficulties with both of the approaches here. In the first measure, there is obviously some subjectivity inherent in our choice of an "ideal grouping;" different researchers may produce different "ideal groupings." However, for rule bases as simple as those used here, people would tend to generate fairly similar "ideal" groups. The second measure of cluster quality avoids subjectivity, but still begs the question of the quality of a grouping; highly stable groupings can still be quite poor. In our view, finding a better way of assessing grouping quality should have as high a priority as that of finding better grouping strategies. Solving either problem would help solve the other.

Measuring Cluster Quality

In many cases, it is easy to see which rules should be grouped together. However, in other cases rules cannot be classified unambiguously, since they relate to several key concepts. For example, in the rules given for the car diagnostic system in Figure 3, we can say that the second rule relates only to battery problems. However, in the first rule, a faulty battery relates to both starter and light problems; there is no best way to group this rule.

Mathematically, we use the following representation. The rule base consists of a set of rules

$$R = \{r_1 \ldots r_n\}$$

and a set of concepts

$$C = \{C_1 \ldots C_l\}.$$

Rule r_i has associated concepts

$$Concepts(r_i) = \{C_{i_1} \ldots C_{i_k}\}$$

where

$$\{C_{i_1} \ldots C_{i_k}\} \subseteq C.$$

We define *unique* rules as those rules that contain only a single concept. Rules that have multiple concepts associated with them are called *ambiguous* rules. The Appendices clarify these views by cross listing each rule with the key concept/concepts contained in it.

To deal with the issue of *ambiguous* rules, we take as our ideal grouping a grouping of the *unique* rules only. This is easy to do and removes some of the subjective component of this approach. Thus, our ideal grouping has the form:

$$I = \{\mathcal{I}_1 \ldots \mathcal{I}_m\},$$

where each $\mathcal{I}_j$ is a disjoint set of *unique* rules.

Now suppose the observed or computed grouping is denoted as:

$$O = \{\mathcal{O}_1 \ldots \mathcal{O}_m\}.$$

Our first step in measuring deviation is to pair each ideal group $\mathcal{I}_j$ with the "closest" observed group,

where we measure closeness using only *unique* rules. Thus, for each index j, let k_j be chosen to maximize:

$$| \mathcal{I}_j \cap \mathcal{O}_{k_j} | .$$

Now we define the deviation $dev(A, B)$ for groups A and B as:

$$dev(A, B) = \{r \epsilon A \mid Concepts(r) \cap Concepts(B) = \phi\},$$

where the concepts in a set of rules are defined as the union of the concepts in the individual rules in the set. The total deviation of the observed groups can then be defined as,

$$tot\_dev = \sum_{i=1}^{m} | dev(\mathcal{O}_i, \bigcup_{j|k_j=i} \mathcal{I}_j) | .$$

The average deviation is thus

$$ave\_dev = \frac{tot\_dev}{n},$$

where n is the total number of rules.

Measuring Stability of Groups

A group formed by the primary-rule clustering algorithm is stable if the replacement of its primary rule by any of the non-primary rules in the group causes no migration of rules to other groups when we rerun the clustering algorithm. Thus, we can measure stability just by counting migration of rules between groups, as each non-primary rule in each group is in turn allowed to assume the role of primary rule for that group. To avoid a combinatorial explosion, we vary the primary rule in only one group at a time. Thus, measuring the stability of a grouping of an n-rule rule base requires only $O(n^3)$ time,

since the primary rule grouping algorithm requires $O(n^2)$ time. Assume the set of primary rules are:

$$\pi = \{p_1 \ldots p_m\}$$

where $p_i = r_{k_i}$. Let G(p_i) be the group computed from primary rule p_i. Then we define the stability of a rule base grouping based on a choice of primary rules π and distance metric d as:

$$Stability(\pi, d) = 1 - \frac{\sum_{i=1}^{m} \frac{1}{|G(p_i)|-1} \sum_{q \epsilon (G(p_i) \setminus p_i)} |\,(G(p_i) \setminus G(q))\,|}{n - m}.$$

Here "\" denotes set differences. That is,

$$A \setminus B = \{a \mid a \in A \text{ and } a \notin B\}.$$

4.2 Analysis of Results

In this subsection we describe the performance of the various distance metrics in grouping the three rule bases using the two clustering algorithms.

Performance of Automatic Rule Clustering

Even if the user is already familiar with the key concepts in a rule base, automatic grouping can provide much insight into the interactions between those concepts. The most interesting aspect of this strategy is the way in which groups form around related sets of concepts. For example, we expected that the rules for battery, starter and engine problems in the car rule base would form independent groups. However, since the physical coupling between these subsystems is reflected in the rule base, these three groups combine under the automatic grouping strategy to form a single large group. Perhaps a more careful choice of distance metric could fix such problems, but we currently have no cure.

Distance Metric	Average deviation for automatic clustering	Average deviation for primary clustering	Stability of primary clustering
d_{all}	0.05	0.04	0.96
d_{df}	0.20	0.84	0.29
d_{ant}	0.05	0.00	0.95

Table 1: Clustering of MMU-FDIR rule base

The effectiveness of the different distance metrics for grouping our three sample rule bases is shown in Table 1 through Table 3. As expected, the d_{ant} distance metric worked well for the car and MMU-FDIR rule base, resulting in average deviation of only 5% from the ideal grouping. Thus, the bulk of the domain knowledge was indeed carried by the antecedents of the rules. The d_{df} distance metric did not do nearly as well for grouping these diagnostic systems.

Because of the chaining nature of rules in the animal rule base, the d_{ant} distance metric did not capture the rule interdependencies well and resulted in 29% deviation from the ideal grouping. However, as expected, the d_{df} distance metric captured this chaining nature very well and resulted in a perfect grouping.

The automatic strategy has done well with the distance metric d_{all} in all three cases. This appears to be a more general metric which works well in the absence of more detailed information on the rule base. Since the deviation between the figures obtained for automatic and primary rule strategies is not very different, the automatic strategy can be proposed as a viable alternative to the primary rule strategy when no selection of primary rules is given.

Performance of Primary Rule Clustering

The primary rule strategy does as well as the automatic strategy in most cases. For our experiments, we did not change the primary rules for the different metrics, because we wanted to judge the metrics under the same conditions. Given the right distance metric and the right primary rules, the primary

Distance Metric	Average deviation for automatic clustering	Average deviation for primary clustering	Stability of primary clustering
d_{all}	0.05	0.05	0.73
d_{df}	0.17	0.25	0.54
d_{ant}	0.05	0.05	0.77

Table 2: Clustering of Car Diagnostic rule base

Distance Metric	Average deviation for automatic clustering	Average deviation for primary clustering	Stability of primary clustering
d_{all}	0.00	0.00	0.73
d_{df}	0.00	0.07	0.55
d_{ant}	0.29	0.21	0.82

Table 3: Clustering of Animal Identification Rule Base

rule clustering gives a perfect grouping for both the animal and MMU-FDIR rule base. For the MMU-FDIR rule base, the perfect grouping was obtained through the d_{ant} metric, and for the animal rule base, through the d_{all} metric. For the car rule base both d_{all} and d_{ant} work well. The data-flow approach does not succeed here for the reasons given in section 2.

Stability figures for almost all the rule bases are consistent with the deviation figures, except in the case of the animal problem. For that case our conclusions are limited due to the small set of rules. We suspect that patterns which are present with both carnivore and ungulate rules, such as dark-spots and black-stripes, interfere with the grouping process. In the MMU-FDIR and car rule base, not only do we get meaningful groups but their stability figures are also consistently good. This means that there is little coupling between the groups produced, as evidenced also by examination of the rule base. Thruster configuration rules, failure recovery rules, and tank/thruster rules can be seen to be structurally different. The data flow metric fails here, because there is no chaining of rules. Each rule's antecedent creates a situation and the consequent asserts a diagnostic message. There is

no relationship, therefore, between the diagnostic performed by one rule, and the situation created by another rule. The primary rule strategy is effective and robust. The price to be paid for this is the need for selecting primary rules.

5 Relationship to Verification and Validation

Given an effective grouping for a rule base, one can approach its verification and validation (V&V) in a manner similar to that used for conventional software. Conventional software yields more easily to verification efforts because it is procedure-driven. Modules can be designed in conventional software, each consisting of a manageable unit with a well-defined interface. These units can then be subjected to unit/integration testing techniques. In expert systems, rules play a role analogous to procedures. However, each rule in an expert system is data-driven, since the presence or absence of data controls the flow of execution. Hence, V&V techniques for expert systems have to view interactions between all pairs of rules. For large expert systems this is quite difficult and can be prohibitively expensive.

Difficulties in verifying expert systems are further compounded by the fact that rapid prototyping and iterative development form a key feature of expert system development. This has led to the development of ad-hoc expert-system design techniques without any software engineering guidelines. However, it is our belief that expert system V&V is not philosophically different from conventional V&V, provided certain software engineering guidelines can be defined for programming them and automated tools developed for verifying them.

Our research efforts have addressed the feasibility of automating the identification of rule groups, in order to decompose the rule base into a number of meaningful units. Each such group can then be viewed as a procedure. A minor extension to our software tool would allow identification of the intra-group and inter-group items for a group of rules, which would be analogous to local variables and parameters for procedures in conventional software. Once a rule base is decomposed into units

with "firewalls," studying the interactions between rules would become more tractable. Perhaps a verification-aid tool could then test the behavior of each such unit under all possible values of inputs[13]. The tool CRSV ¶ [14], for CLIPS is already equipped to provide information on the allowed values for each field.

The grouping of rule bases can play an important role in verification and validation of flight-critical systems. In such systems one needs to identify critical regions, assert various criticality levels[1] for them, and test such regions both analytically and exhaustively. If one is able to isolate the group of rules that deal with the critical features of the problem domain, certain safety properties of the system can be verified. Knowledge of the function of a group of rules would allow us to choose the distribution of inputs in such a way that typical situations where functioning of the system is critical could be studied in isolation [15]. Moreover, if support existed for specifying what rules should not get fired under certain circumstances, backward flow analysis techniques [16] could be used to locate critical paths. An additional advantage of modularization would be the identification of modules and data items that are necessary in a degraded (fail-soft) processing mode. Validating such modules is clearly critical to confidence in the reliability of the software.

6 Conclusions

In this paper we have described semi-automatic grouping of rule-based systems, using two different approaches. The first was completely automatic, while the second clustered rules around primary rules selected by the user. We also defined three different distance metrics for our grouping algorithms, each capturing different kinds of information in the rules. We tested our ideas on three sample rule bases, and attempted to cluster them using both clustering algorithms, and trying all three metrics for each algorithm.

¶CLIPS Cross Reference Style Analysis and Verification Tool

Our experiments have shown that effective grouping can be achieved with both the automatic and primary rule clustering strategies. The primary rule approach works very well, if a reasonable set of primary rules can be easily identified. In the absence of a reasonable primary rule set, the automatic approach could serve as a useful fall-back strategy.

The choice of metric is also important. In the absence of specific knowledge about the nature of the rule base, a safe approach is to use the distance metric taking all patterns in each rule into account. However, one can clearly do better by using a metric tuned to the rule base, as our results have demonstrated.

There are a number of minor problems with our approach to grouping. For example, attributes irrelevant to the grouping process tend to distort the clustering of groups. In general, the effectiveness of our approach is fundamentally limited by its reliance on syntactic pattern-matching and cannot take into account the semantics of the patterns. As noted by Michalski and Stepp [17], "a configuration of objects forms a class only if it can be closely circumscribed by a conjunctive concept involving relations on selected object attributes." Moreover, a pattern-matching approach cannot capture the "Gestalt properties" of rule clusters. In other words, the meaning of the whole is generally greater than the sum of its parts. A more sophisticated approach is required. Our measurement of deviation of computed groups from ideal groups contains ideas which could lead to grouping strategies reflecting Michalski and Stepp's insight; we hope to pursue this in the future.

However, even with these limitations, the techniques we have developed seem effective at grouping rule bases, not only as judged by our stability measure, but also when judged in terms of the way one would intuitively group these rule bases. Therefore, we feel that these techniques could be very useful both in the validation phase and during maintenance of rule bases.

Acknowledgments

We would like to thank Paul Miner and Sally Johnson of NASA, and Shahid Bokhari and John V. Rosendale of ICASE for their helpful suggestions.

References

[1] S. C. Johnson. Validation of highly reliable, real-time knowledge-based systems. In *SOAR 88 Workshop on Automation and Robotics*, July 1988.

[2] S. Lindell. Keyword cluster algorithm for expert system rule bases. Technical Report SD-TR-87-36, The Aerospace Corporation, El Segundo, CA., June 1987.

[3] K. Lindenmayer, S. Vick, and D. Rosenthal. Maintaining an expert system for the Hubble space telescope ground support. In *Proceedings of the Goddard Conference on Space Applications of Artificial Intelligence and Robotics*, pages 1–13, May 1987.

[4] R. J. K. Jacob and J. N. Froscher. Developing a software engineering methodology for knowledge-based systems. Technical Report 9019, Naval Research Laboratory, Washington, D.C., December 1986.

[5] R. J. K. Jacob and J. N. Froscher. A software engineering methodology for rule-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):173–189, June 1990.

[6] M. Mehrotra and S. C. Johnson. Rule groupings in expert systems. In *Proceedings of the First CLIPS Users Group Conference*, Aug 1990.

[7] M. Mehrotra and S. C. Johnson. Importance of rule groupings in verification of expert systems. In *Notes for the AAAI-90 Workshop on Verification, Validation and Testing of Knowledge-Based Systems*, July 1990.

[8] Knowledge Requirements for the Onboard Navigation Console Expert/Trainer System. Technical Report JSC-22657, NASA, Lyndon B. Johnson Space Center, Houston, TX., September 1988.

[9] D. G. Lawler and L. J. F. Williams. MMU FDIR Automation Task. Technical Report NAS9-17650, McDonnell Douglas Astronautics - Engineering Services, Houston, TX., February 1988.

[10] J. Giarratano and G. Riley. *Expert Systems Principles and Programming.* PWS-KENT Publishing Company, 1989.

[11] J. C. Giarratano. CLIPS User's Guide. Technical report, Artificial Intelligence Center, NASA, Lyndon B. Johnson Space Center, Houston, TX., June 1989.

[12] P. H. Winston. *Artificial Intelligence.* Addison Wesley Publishing Company, 1979.

[13] C. Culbert and R. T. Savely. Expert system verification and validation. In *Proceedings of the Validation and Testing Knowledge-Based Systems Workshop*, August 1988.

[14] CLIPS Reference Manual. Technical Report JSC-22948, Artificial Intelligence Center, NASA, Lyndon B. Johnson Space Center, Houston, TX., June 1989.

[15] D. L. Parnas, J. van Schouwen, and S. Po Kwan. Evaluation of safety-critical software. *Communications of the ACM*, 33(6):636–648, June 1990.

[16] N. G. Leveson. Safety-critical software development. In T. Anderson, editor, *Safe & Secure Computing Systems*, chapter 9, pages 155–162. Blackwell Scientific Publications, 1989.

[17] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning - An artificial Intelligence Approach*, chapter 11, pages 331–362. Tioga Publishing Company.

Appendix A - ANIMAL IDENTIFICATION RULE BASE

```
(defrule r1 ( animal_has_hair) => ( animal_is_a_mammal))

(defrule r2 ( animal_gives_milk) => ( animal_is_a_mammal))

(defrule r3 ( animal_has_feather) => ( animal_is_a_bird))

(defrule r4 ( animal_can_fly)
 ( animal_lays_eggs) => ( animal_is_a_bird))

(defrule r5 ( animal_eats_meat) => ( animal_is_a_carn))

(defrule r6 ( animal_has_pointed_teeth)
(animal_has_claws_forward_eyes)
     => ( animal_is_a_carnivore))

(defrule r7 ( animal_is_a_mammal)
( animal_has_hoofs)
=> ( animal_is_a_ungulate))

(defrule r8 ( animal_is_a_mammal)
( animal_is_a_carnivore)
( animal_has_tawny_color)
( animal_has_darkspots)
=>  (animal_is_a_cheetah))

(defrule r9 ( animal_is_a_mammal)
( animal_is_a_carnivore)
( animal_has_tawny_color)
( animal_has_blackstripes)
=> (animal_is_a_tiger))

(defrule r10 ( animal_is_a_ungulate)
( animal_has_long_neck)
( animal_has_long_legs)
( animal_has_darkspots)
=> (animal_is_a_giraffe))

(defrule r11 (animal_is_a_ungulate)
(animal_has_blackstripes)
=> (animal_is_a_zebra))

(defrule r12 (animal_is_a bird)
( animal_can_not_fly)
( animal_has_long_neck)
( animal_has_long_legs)
( animal_has_black_and_white_color)
```

```
=> (animal_is_a_ostrich))

(defrule r13 (animal_is_a_bird)
( animal_can_not_fly)
( animal_can_swim)
( animal_has_black_and_white_color)
=> (animal_is_a_penguin))

(defrule r14 (animal_is_a_bird)
(animal_can_fly_well)
=> (animal_is_a_albatross))

--------------------------------------------------------------------------------

Key Concepts carried by each rule :

1 mammal
2 mammal
3 bird
4 bird
5 carn
6 carn
7 mammal,ungulate
8 mammal,carn
9 mammal,carn
10 ungulate
11 ungulate
12 bird
13 bird
14 bird

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
PRIMARY RULE CLUSTERING
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

The primary-rules are:
1 mammal
5 carn
12 bird

**** DISTANCE METRIC => TOTAL ****

rules belonging to the group 1 are:
1 mammal
2 mammal
7 mammal,ungulate
```

```
11 ungulate

rules belonging to the group 5 are:
5 carn
6 carn
8 mammal,carn
9 mammal,carn

rules belonging to the group 12 are:
12 bird
3 bird
4 bird
10 ungulate
13 bird
14 bird

 Average stability: 0.727273

**** DISTANCE METRIC => DATA_FLOW ****

rules belonging to the group 1 are:
1 mammal
2 mammal
6 carn
7 mammal,ungulate
10 ungulate
11 ungulate

rules belonging to the group 5 are:
5 carn
8 mammal,carn
9 mammal,carn

rules belonging to the group 12 are:
12 bird
3 bird
4 bird
13 bird
14 bird

 Average stability: 0.545455

**** DISTANCE METRIC => LEFT_ONLY ****

rules belonging to the group 1 are:
1 mammal
2 mammal
3 bird
```

```
4 bird
6 carn

rules belonging to the group 5 are:
5 carn

rules belonging to the group 12 are:
12 bird
7 mammal,ungulate
8 mammal,carn
9 mammal,carn
10 ungulate
11 ungulate
13 bird
14 bird

 Average stability: 0.818182

--------------------------------------------------------------------------------
AUTOMATIC RULE CLUSTERING
--------------------------------------------------------------------------------

ANIMAL : TOTAL : A : 3 groups

**** DISTANCE METRIC => TOTAL ****

The global stopping criteria is 3 groups:

rules belonging to the group 1 are:
1 mammal
2 mammal
7 mammal,ungulate
8 mammal,carn
9 mammal,carn
11 ungulate

rules belonging to the group 2 are:
3 bird
4 bird
14 bird
10 ungulate
12 bird
13 bird

rules belonging to the group 3 are:
5 carn
6 carn
```

```
--------------------------------------------------------------------------

ANIMAL : LEFT_ONLY : A : 3 groups

**** DISTANCE METRIC => LEFT_ONLY ****

The global stopping criteria is 3 groups:

rules belonging to the group 1 are:
1 mammal
2 mammal
3 bird
4 bird
5 carn

rules belonging to the group 2 are:
6 carn

rules belonging to the group 3 are:
7 mammal,ungulate
8 mammal,carn
9 mammal,carn
11 ungulate
10 ungulate
12 bird
13 bird
14 bird

--------------------------------------------------------------------------

ANIMAL : DATA_FLOW : A : 3 groups

**** DISTANCE METRIC => DATA_FLOW ****

The global stopping criteria is 3 groups:

rules belonging to the group 1 are:
1 mammal
7 mammal,ungulate
2 mammal
8 mammal,carn
11 ungulate
10 ungulate

rules belonging to the group 2 are:
3 bird
14 bird
4 bird
```

```
13 bird
12 bird

rules belonging to the group 3 are:
5 carn
9 mammal,carn
6 carn
```

Appendix B - CAR DIAGNOSTICS RULE BASE

```
(defrule starter_ok ;rule_no 1
(starter cranks_engine yes)
(lights dim slightly)
=>
(assert (battery problems no))
(assert (engine is_tight no))
(printout t "battery strength O.K." crlf)
(printout t "battery connections O.K." crlf)
(printout t "speed looks good " crlf)
)

(defrule battery_not_ok ;rule_no 2
(starter cranks_engine yes)
(lights dim considerably)
=>
(assert (battery problems yes))
(assert (engine is_tight yes))
)

(defrule engine_misfires ;rule_no 3
(battery problems no)
(engine is_tight no)
(engine misfires constantly)
=>
(printout t "short one plug at a time; try to locate weak or
misfiring cylinder and firing cylinder" crlf)
(printout t "either locate misfiring cylinder and enter" crlf)
(printout t "cylinder <no> misfiring" crlf)
(printout t "and cylinder <no> isfiring" crlf)
(printout t " or enter engine misfires idle_speed|high_speed" crlf)
(assert(cylinder misfiring locate ))
)

(defrule engine_misfires2 ;rule_no 4
(battery problems no)
(engine is_tight no)
(engine runs unevenly)
=>
(printout t "short one plug at a time; try to locate weak or
misfiring cylinder and firing cylinder" crlf)
(printout t "either locate misfiring cylinder and enter" crlf)
(printout t "cylinder <no> misfiring" crlf)
(printout t "and cylinder <no> isfiring" crlf)
(printout t " or enter engine misfires idle_speed|high_speed" crlf)
(assert(cylinder misfiring locate ))
```

```
)

(defrule to_locate_misfiring_cylinder ;rule_no 5
?z <- (cylinder misfiring locate)
(cylinder misfiring ?x)
(cylinder isfiring ?y)
=>
(retract ?z)
)

(defrule located_misfiring_cylinder ;rule_no 6
?a <- (cylinder misfiring ?x)
?b <- (cylinder isfiring ?y)
=>
(retract ?a ?b)
(printout t "reverse spark_plugs on" ?x " " ?y "cyliders" crlf)
(printout t "note change of spark_plugs" crlf)
(assert(spark_plug change note))
)

(defrule no_change_in_spark_plugs ;rule_no 7
?z <- (spark_plug change note)
(spark_plug change no)
=>
(assert (cylinder problem1 yes))
(assert (distributor problem1 yes))
(retract ?z)
)


(defrule distributor_problem1 ; no high voltage to spark_plug ;rule_no 8
?z <- (distributor problem1 yes)
=>
(printout t "leaking high tension wires" crlf )
(printout t "defective distributor cap" crlf )
)

(defrule cylinder_problems  ;compression not good enough for ignition ;rule_no 9
?z <- (cylinder problem1 yes)
=>
(retract ?z)
(printout t "worn valve stem or guide" crlf )
(printout t "poor compression caused by" crlf)
(printout t "sticking, warped, burnt or broken valve" crlf)
(printout t "valve tappets set too close " crlf)
(printout t "valve tappets sticking " crlf)
(printout t "valve spring weak or broken " crlf)
(printout t "valve seat cracked " crlf)
```

```
(printout t "valve seat insert loose " crlf)
(printout t "valve seat warped " crlf)
(printout t "head gasket defective " crlf)
(printout t "cylinder rings broken, stuck or weak " crlf)
(printout t "cylinder valve scoured " crlf)
(printout t "piston broken " crlf)
(printout t "cylinder head or block warped " crlf)
(printout t "cylinder head or block cracked " crlf)
)

(defrule spark_plug_problems_persist ;rule_no 10
?z <- (spark_plug change note)
(spark_plug change yes)
=>
(retract ?z)
(assert(spark_plug problem1 yes))
)

(defrule change_spark_plugs ;rule_no 11
?z <- (spark_plug problem1 yes)
=>
(retract ?z)
(printout t "try cleaning spark plug " crlf)
(printout t "try adjusting spark plug gap" crlf)
(printout t "replace with new plug having proper heat range " crlf)
(printout t "appearance of porcelian at lower end shows if plug is correct " crlf)
(assert(spark_plug replace yes))
)

(defrule spark_plug_replace ;rule_no 12
?z <- (spark_plug replace yes)
=>
(retract ?z)
(printout t "note the color of spark_plug in <spark_plug porcelian_color x>" crlf)
(assert (spark_plug porcelian_color note ))
)

(defrule to_note_spark_plug_color ;rule_no 13
?z <- (spark_plug porcelian_color note )
(spark_plug porcelian_color ?color)
=>
(retract ?z)
)

(defrule spark_plug_choose_cold ;rule_no 14
(spark_plug porcelian_color ashwhite)
=>
(printout t "plug too hot; use colder plug" crlf)
```

```
)

(defrule spark_plug_choose_same ;rule_no 15
(spark_plug porcelian_color light_brown)
=>
(printout t "plug just right; replace with same type" crlf)
)

(defrule spark_plug_choose_hot_1 ;rule_no 16
(spark_plug porcelian_color black)
=>
(printout t "plug too cold; use hotter plug" crlf)
)

(defrule spark_plug_choose_hot_2 ;rule_no 17
(spark_plug porcelian_color oily)
=>
(printout t "plug too cold; use hotter plug" crlf)
)

(defrule misfiring_at_idle_speed ;rule_no 18
?z <- (cylinder misfiring locate)
(engine misfires idle_speed)
=>
(retract ?z)
(assert(spark_plug problem2 yes))
(assert(ignition_switch problems yes))
(assert(distributor problem2 yes))
(assert(carburetor problem1 yes))
(assert(cylinder problem2 yes))
)

(defrule spark_plug_problem2 ;rule_no 19
?z <- (spark_plug problem2 yes)
=>
(retract ?z)
(printout t "spark_plug gaps too wide" crlf crlf)
(printout t "spark_plugs defective" crlf)
)

(defrule ignition_switch_problems ;rule_no 20
?z <- (ignition_switch problems yes)
=>
(retract ?z)
(printout t "ignition_switch defective" crlf)
)

(defrule distributor_problem2 ;rule_no 21
```

```
?z <- (distributor problem2 yes)
=>
(retract ?z)
(printout t "high tension wires defective " crlf)
(printout t "distributor shaft_bushings worn " crlf)
(printout t "distributor rotor defective " crlf)
(printout t "condenser defective" crlf)
(printout t "breaker points defective" crlf)
(printout t "breaker arm sticking" crlf)
(printout t "breaker point gap incorrect" crlf)
(printout t "spark_advance too far" crlf)
)

(defrule carburetor_problem1 ;rule_no 22
?z <- (carburetor problem1 yes)
=>
(retract ?z)
(printout t "carburetor adjustments incorrect " crlf)
(printout t "carburetor float incorrect " crlf)
(printout t "carburetor system has_dirt" crlf)
(printout t "carburetor system has_water" crlf)
(printout t "carburetor system is_vapor_locked" crlf)
(printout t "carburetor manifold_intake has_air_leak" crlf)
)

(defrule cylinder_problem2   ;compression not good enough to run well;rule_no 23
?z <- (cylinder problem2 yes)
=>
(retract ?z)
(printout t "valve_intake has_air_leak" crlf)
(printout t "valves sticking or open" crlf)
(printout t "valves burnt or warped" crlf)
(printout t "valves broken" crlf)
(printout t "valve springs weak" crlf)
(printout t "valve springs broken" crlf)
(printout t "valve_tappet clearance incorrect" crlf)
(printout t "valve seat cracked" crlf)
(printout t "valve seat warped" crlf)
(printout t "valve seat_insert loose" crlf)
(printout t "valve lifter sticking" crlf)
(printout t "piston rings sticking" crlf)
(printout t "piston rings broken" crlf)
(printout t "piston broken" crlf)
(printout t "head_gasket defective" crlf)
(printout t "cylinder walls scoured " crlf)
(printout t "cylinder_head warped" crlf)
)
```

```
(defrule misfiring_at_high_speed ;rule_no 24
?y <- (cylinder misfiring locate )
?z <- (engine misfires high_speed)
=>
(retract ?y)
(retract ?z)
(assert(spark_plug problem3 yes))
(assert(distributor problem3 yes))
(assert(cylinder problem3 yes))
)

(defrule spark_plugs_problem3 ;rule_no 25
?z <- (spark_plug problem3 yes)
=>
(retract ?z)
(printout t "spark_plugs wrong_type" crlf)
)

(defrule distributor_problem3 ;rule_no 26
?z <- (distributor problem3 yes)
=>
(retract ?z)
(printout t "breaker_arm springs weak" crlf)
(printout t "breaker_points too wide" crlf)
)

(defrule cylinder_problem3 ;rule_no 27
?z <- (cylinder problem3 yes)
=>
(retract ?z)
(printout t "cylinder has_excessive_carbon" crlf)
(printout t "valve springs weak" crlf)
)

(defrule starter_not_ok_1 ;rule_no 28
(starter cranks_engine no)
(lights dim not_at_all) ; lights stay bright
=>
(printout t "starter_switch has_open_circuit" crlf)
(printout t "starter_motor has_open_circuit" crlf)
(printout t "starter_brushes has_no_contact_with armature or rotor" crlf)
)

(defrule starter_not_ok_2 ;rule_no 29
(starter cranks_engine no)
(lights dim slightly)
=>
(printout t "starter may not engage with engine" crlf)
```

```
(printout t "starter switch resistance too_high" crlf)
)


(defrule starter_not_ok_3_1 ;rule_no 30
(starter cranks_engine no)
(lights dim considerably)
=>
(assert(battery problems yes))
(assert(starter problems yes))
(assert(engine is_tight yes ))
)

(defrule starter_not_ok_3_2 ;rule_no 31
(starter cranks_engine no)
(lights dim totally) ; lights go out
=>
(assert(battery problems yes))
(assert(starter problems yes))
(assert(engine is_tight yes ))
)

(defrule battery_problems ;rule_no 32
?z <- (battery problems yes)
=>
(retract ?z)
(printout t "battery strength weak" crlf)
(printout t "battery terminals are loose or corroded" crlf)
(printout t "battery cable loose or defective" crlf)
(printout t "battery discharged" crlf)
(printout t "battery connections poor" crlf)
)

(defrule starter_problems ;rule_no 33
?z <- (starter problems yes)
=>
(retract ?z)
(printout t "starter binds shorted" crlf)
)

(defrule engine_tight_problem ;rule_no 34
?z <- (engine is_tight yes)
=>
(retract ?z)
(printout t "engine is too tight" crlf)
)

(defrule engine_lacks_power ;rule_no 35
```

```
(battery problems no)
(engine is_tight no)
(engine power lacking)
=>
(assert(distributor problem4 yes))
(assert(carburetor problem2 yes))
(assert(cylinder problem4 yes))
(assert(exhaust problems yes))
(assert(engine problems yes))
(assert(drive_train problems yes))
(assert(fuel problem1 yes))
)

(defrule distributor_problem4 ;rule_no 36
?z <- (distributor problem4 yes)
=>
(retract ?z)
(printout t "ignition improperly timed" crlf)
(printout t "wrong routing of high tension wires" crlf)
(printout t "ignition points not properly synchronized" crlf)
(printout t "automatic advance not operating properly" crlf)
(printout t "vacuum spark control not operating properly" crlf)
)

(defrule carburetor_problem2 ;rule_no 37
?z <- (carburetor problem2 yes)
=>
(retract ?z)
(printout t "carburetor adjustments incorrect" crlf)
(printout t "air_cleaner clogged" crlf)
)

(defrule cylinder_problem4 ;rule_no 38
?z <- (cylinder problem4 yes)
=>
(retract ?z)
(printout t "valve timing incorrect" crlf)
)

(defrule exhaust_problems ;rule_no 39
?z <- (exhaust problems yes)
=>
(retract ?z)
(printout t "muffler clogged" crlf)
(printout t "exhaust_pipe dented" crlf)
)
```

```
(defrule engine_problems ;rule_no 40
?z <- (engine problems yes)
=>
(retract ?z)
(printout t "engine overheating" crlf)
(printout t "engine friction excessive" crlf)
)

(defrule drive_train_problem ;rule_no 41
?z <- (drive_train problems yes)
=>
(retract ?z)
(printout t "clutch slipping" crlf)
(printout t "chassis has_drag which retards free running of car" crlf)
)

(defrule fuel_problem1 ;rule_no 42
?z <- (fuel problem1 yes)
=>
(retract ?z)
(printout t "vapor lock" crlf)
)

(defrule engine_will_not_run ;rule_no 43
(battery problems no)
(engine is_tight no)
(engine runs not_at_all)
=>
(printout t "remove spark_plug_wire and hold near engine while cranking")
(assert(spark strength type))
)

(defrule weak_spark ;rule_no 44
?z <- (spark strength type)
(spark strength weak)
=>
(retract ?z)
(assert(distributor problem5 yes))
)


(defrule distributor_problem5 ;rule_no 45
?z <- (distributor problem5 yes)
=>
(retract ?z)
(printout t "distributor rotor defective" crlf)
(printout t "distributor cap defective" crlf)
(printout t "rotor brush broken" crlf)
```

```
(printout t "coil distributor wet" crlf)
(printout t "points dirty or pitted" crlf)
(printout t "electrical connections poor" crlf)
(printout t "high tension wires defective" crlf)
(printout t "high tension wires wet" crlf)
(printout t "coil defective" crlf)
(printout t "condenser defective" crlf)
)

(defrule no_spark ;rule_no 46
?z <- (spark strength type)
(spark strength none)
=>
(retract ?z)
(assert(ammeter reading note ))
)

(defrule ammeter_reading ;rule_no 47
?z <- (ammeter reading note)
(ammeter reading ?x)
(ammeter needle ?y)
=>
(retract ?z)
)

(defrule ammeter_none ;rule_no 48
(ammeter reading none)
(ammeter needle steady)
=>
(assert(distributor problem6 yes))
)

(defrule distributor_problem_6 ;rule_no 49
?z <- (distributor problem6 yes)
=>
(retract ?z)
(printout t "points are not closing due to" crlf)
(printout t "points dirty, pitted or burnt" crlf)
(printout t "switch defective" crlf)
(printout t "coil_winding open" crlf)
(printout t "primary_wire open" crlf)
(printout t "connections loose" crlf)
)

(defrule ammeter_normal ;rule_no 50
(ammeter reading normal)
(ammeter needle unsteady)
=>
```

```
(assert(distributor problem7 yes))
)

(defrule distributor_problem7 ;rule_no 51
?z <- (distributor problem7 yes)
=>
(printout t "distributor_rotor defective" crlf)
(printout t "distributor_cap defective" crlf)
(printout t "distributor_coil wet" crlf)
(printout t "high_tension_wire from coil to distributor open or grounded" crlf)
(printout t "coil defective" crlf)
(printout t "condenser defective" crlf)
(printout t "high_tension_wires wet" crlf)
)

(defrule ammeter_discharged ;rule_no 52
(ammeter reading discharge)
(ammeter needle steady)
=>
(assert(distributor problem8 yes))
)

(defrule distributor_problem8 ;rule_no 53
?z <- (distributor problem8 yes)
=>
(retract ?z)
(printout t "breaker points not opening due to" crlf)
(printout t "condenser shorted " crlf)
(printout t "contact_arm grounded" crlf)
(printout t "primary_coil_winding shorted" crlf)
(printout t "primary_circuit  shorted" crlf)
)

(defrule good_spark ;rule_no 54
?z <- (spark strength type)
(spark strength good)
=>
(retract ?z)
(printout t "check fuel supply to see if there is gas in carburetor" crlf)
(assert(fuel supply check ))
)

(defrule check_gas ;rule_no 55
?z <- (fuel supply check )
(fuel supply ?x)
=>
(retract ?z)
)
```

```
(defrule fuel_supply_good ;rule_no 56
(fuel supply good)
=>
(assert(carburetor problem3 yes))
(assert(cylinder problem5 yes))
)

(defrule cylinder_problem5 ;rule_no 57
?z <- (cylinder problem5 yes)
=>
(retract ?z)
(printout t "cylinders have_water_leaks" crlf)
)

(defrule carburetor_problem3 ;rule_no 58
?z <- (carburetor problem3 yes)
=>
(retract ?z)
(printout t "carburetor flooded" crlf)
(printout t "carburetor has_dirt or has_water" crlf)
(printout t "choke not_operating" crlf)
)

(defrule fuel_supply_empty ;rule_no 59
(fuel supply none)
=>
(assert (fuel problem2 yes))
)

(defrule fuel_problem2 ;rule_no 60
?z <-  (fuel problem2 yes)
=>
(retract ?z)
(printout t "fuel_lines clogged" crlf)
(printout t "fuel_filter clogged" crlf)
(printout t "tank_cap has_no_vent" crlf)
(printout t "fuel_supply_unit defective" crlf)
(printout t "tank_line has_air_leak" crlf)
)

--------------------------------------------------------------------------
Key concepts carried by the rules
--------------------------------------------------------------------------

1 battery,starter,engin
2 battery,starter,engine
3 battery,engine,cylinder
```

```
4 battery,engine,cylinder
5 cylinder
6 cylinder
7 cylinder,sp_plug,distributor
8 distributor
9 cylinder
10 sp_plug
11 sp_plug
12 sp_plug
13 sp_plug
14 sp_plug
15 sp_plug
16 sp_plug
17 sp_plug
18 engine,cylinder,sp_plug,ignition,carburetor,distributor
19 sp_plug
20 ignition
21 distributor
22 carburetor
23 cylinder
24 engine,cylinder,sp_plug,distributor
25 sp_plug
26 distributor
27 cylinder
28 starter+l
29 starter+l
30 starter,battery,engine
31 starter,battery,engine
32 battery
33 starter
34 engine
35 battery,engine,carburetor,distributor,exhaust,drive-train
36 distributor
37 carburetor
38 cylinder
39 exhaust
40 engine
41 drive-train
42 fuel
43 battery,engine
44 distributor+s
45 distributor
46 distributor+s
47 ammeter
48 distributor,ammeter
49 distributor
50 distributor,ammeter
51 distributor
```

```
52 distributor,ammeter
53 distributor
54 fuel+s
55 fuel
56 fuel,carburetor,cylinder
57 cylinder
58 carburetor
59 ammeter
60 ammeter

------------------------------------------------------------------------------
------------------------------------------------------------------------------
PRIMARY RULE CLUSTERING
------------------------------------------------------------------------------
------------------------------------------------------------------------------

The primary-rules are:
32 battery
33 starter
40 engine
5 cylinder
10 sp_plug
20 ignition
22 carburetor
8 distributor
59 ammeter
47 ammeter
39 exhaust
41 drive-train

**** DISTANCE METRIC => TOTAL ****

rules belonging to the group 32 are:
32 battery
1 battery,starter,engine
2 battery,starter,engine
3 battery,engine,cylinder
4 battery,engine,cylinder
43 battery,engine

rules belonging to the group 33 are:
33 starter
28 starter+1
29 starter+1
30 starter,battery,engine
31 starter,battery,engine

rules belonging to the group 40 are:
```

```
40 engine
34 engine
35 battery,engine,carburetor,distributor,exhaust,drive-train

rules belonging to the group 5 are:
5 cylinder
6 cylinder
18 engine,cylinder,sp_plug,ignition,carburetor,distributor
23 cylinder
24 engine,cylinder,sp_plug,distributor
27 cylinder
38 cylinder

rules belonging to the group 10 are:
10 sp_plug
7 cylinder,sp_plug,distributor
9 cylinder
11 sp_plug
12 sp_plug
13 sp_plug
14 sp_plug
15 sp_plug
16 sp_plug
17 sp_plug
19 sp_plug
25 sp_plug

rules belonging to the group 20 are:
20 ignition

rules belonging to the group 22 are:
22 carburetor
37 carburetor
58 carburetor

rules belonging to the group 8 are:
8 distributor
21 distributor
26 distributor
36 distributor
44 distributor+s
45 distributor
49 distributor
51 distributor
53 distributor

rules belonging to the group 59 are:
59 ammeter
```

```
42 fuel
54 fuel+s
55 fuel
56 fuel,carburetor,cylinder
57 cylinder
60 ammeter

rules belonging to the group 47 are:
47 ammeter
46 distributor+s
48 distributor,ammeter
50 distributor,ammeter
52 distributor,ammeter

rules belonging to the group 39 are:
39 exhaust

rules belonging to the group 41 are:
41 drive-train

 Average stability: 0.729167

**** DISTANCE METRIC => DATA_FLOW ****

rules belonging to the group 32 are:
32 battery
1 battery,starter,engine
2 battery,starter,engine
34 engine
43 battery,engine
44 distributor+s
28 starter+1
29 starter+1

rules belonging to the group 33 are:
33 starter
30 starter,battery,engine
31 starter,battery,engine

rules belonging to the group 40 are:
40 engine
37 carburetor
42 fuel

rules belonging to the group 5 are:
5 cylinder
3 battery,engine,cylinder
4 battery,engine,cylinder
```

```
rules belonging to the group 10 are:
10 sp_plug
6 cylinder
11 sp_plug
12 sp_plug
13 sp_plug
14 sp_plug
15 sp_plug
16 sp_plug
17 sp_plug
19 sp_plug
24 engine,cylinder,sp_plug,distributor
25 sp_plug

rules belonging to the group 20 are:
20 ignition
18 engine,cylinder,sp_plug,ignition,carburetor,distributor

rules belonging to the group 22 are:
22 carburetor

rules belonging to the group 8 are:
8 distributor
7 cylinder,sp_plug,distributor
9 cylinder
21 distributor
23 cylinder
26 distributor
27 cylinder
36 distributor
38 cylinder
45 distributor
49 distributor
51 distributor
53 distributor
56 fuel,carburetor,cylinder
57 cylinder
58 carburetor

rules belonging to the group 59 are:
59 ammeter
54 fuel+s
55 fuel
60 ammeter

rules belonging to the group 47 are:
47 ammeter
```

```
46 distributor+s
48 distributor,ammeter
50 distributor,ammeter
52 distributor,ammeter

rules belonging to the group 39 are:
39 exhaust

rules belonging to the group 41 are:
41 drive-train
35 battery,engine,carburetor,distributor,exhaust,drive-train

 Average stability: 0.541667

**** DISTANCE METRIC => LEFT_ONLY ****

rules belonging to the group 32 are:
32 battery
3 battery,engine,cylinder
4 battery,engine,cylinder
35 battery,engine,carburetor,distributor,exhaust,drive-train
43 battery,engine
44 distributor+s
46 distributor+s
54 fuel+s

rules belonging to the group 33 are:
33 starter
1 battery,starter,engine
2 battery,starter,engine
28 starter+l
29 starter+l
30 starter,battery,engine
31 starter,battery,engine

rules belonging to the group 40 are:
40 engine
34 engine

rules belonging to the group 5 are:
5 cylinder
6 cylinder
9 cylinder
18 engine,cylinder,sp_plug,ignition,carburetor,distributor
23 cylinder
24 engine,cylinder,sp_plug,distributor
27 cylinder
38 cylinder
```

```
57 cylinder

rules belonging to the group 10 are:
10 sp_plug
7 cylinder,sp_plug,distributor
11 sp_plug
12 sp_plug
13 sp_plug
14 sp_plug
15 sp_plug
16 sp_plug
17 sp_plug
19 sp_plug
25 sp_plug

rules belonging to the group 20 are:
20 ignition

rules belonging to the group 22 are:
22 carburetor
37 carburetor
58 carburetor

rules belonging to the group 8 are:
8 distributor
21 distributor
26 distributor
36 distributor
45 distributor
49 distributor
51 distributor
53 distributor

rules belonging to the group 59 are:
59 ammeter
42 fuel
55 fuel
56 fuel,carburetor,cylinder
60 ammeter

rules belonging to the group 47 are:
47 ammeter
48 distributor,ammeter
50 distributor,ammeter
52 distributor,ammeter

rules belonging to the group 39 are:
39 exhaust
```

```
rules belonging to the group 41 are:
41 drive-train

 Average stability: 0.770833

--------------------------------------------------------------------------------
AUTOMATIC RULE CLUSTERING
--------------------------------------------------------------------------------
CAR : TOTAL : V : 12 groups

**** DISTANCE METRIC => TOTAL ****

The global stopping criteria is 12 groups:

rules belonging to the group 1 are:
1 battery,starter,engine
2 battery,starter,engine
30 starter,battery,engine
31 starter,battery,engine
28 starter+1
29 starter+1
33 starter
3 battery,engine,cylinder
4 battery,engine,cylinder
43 battery,engine
35 battery,engine,carburetor,distributor,exhaust,drive-train

rules belonging to the group 2 are:
5 cylinder
6 cylinder
7 cylinder,sp_plug,distributor
18 engine,cylinder,sp_plug,ignition,carburetor,distributor
24 engine,cylinder,sp_plug,distributor
10 sp_plug
11 sp_plug
12 sp_plug
19 sp_plug
25 sp_plug
13 sp_plug
14 sp_plug
15 sp_plug
16 sp_plug
17 sp_plug

rules belonging to the group 3 are:
8 distributor
21 distributor
```

```
26 distributor
36 distributor
45 distributor
49 distributor
51 distributor
53 distributor
47 ammeter
48 distributor,ammeter
52 distributor,ammeter
50 distributor,ammeter

rules belonging to the group 4 are:
9 cylinder
23 cylinder
27 cylinder
38 cylinder
57 cylinder

rules belonging to the group 5 are:
20 ignition

rules belonging to the group 6 are:
22 carburetor
37 carburetor
58 carburetor
56 fuel,carburetor,cylinder

rules belonging to the group 7 are:
32 battery

rules belonging to the group 8 are:
34 engine
40 engine

rules belonging to the group 9 are:
39 exhaust

rules belonging to the group 10 are:
41 drive-train

rules belonging to the group 11 are:
42 fuel
60 ammeter
59 ammeter
55 fuel

rules belonging to the group 12 are:
44 distributor+s
```

```
46 distributor+s
54 fuel+s

------------------------------------------------------------------------

CAR : LEFT_ONLY : V : 12 groups

**** DISTANCE METRIC => LEFT_ONLY ****

The global stopping criteria is 12 groups:

rules belonging to the group 1 are:
1 battery,starter,engine
2 battery,starter,engine
28 starter+1
29 starter+1
30 starter,battery,engine
31 starter,battery,engine
33 starter

rules belonging to the group 2 are:
3 battery,engine,cylinder
4 battery,engine,cylinder
43 battery,engine
35 battery,engine,carburetor,distributor,exhaust,drive-train
32 battery
34 engine
40 engine

rules belonging to the group 3 are:
5 cylinder
6 cylinder
18 engine,cylinder,sp_plug,ignition,carburetor,distributor
24 engine,cylinder,sp_plug,distributor
9 cylinder
23 cylinder
27 cylinder
38 cylinder
57 cylinder

rules belonging to the group 4 are:
7 cylinder,sp_plug,distributor
10 sp_plug
13 sp_plug
11 sp_plug
12 sp_plug
19 sp_plug
25 sp_plug
```

```
 14 sp_plug
 15 sp_plug
 16 sp_plug
 17 sp_plug

 rules belonging to the group 5 are:
 8 distributor
 21 distributor
 26 distributor
 36 distributor
 45 distributor
 49 distributor
 51 distributor
 53 distributor

 rules belonging to the group 6 are:
 20 ignition

 rules belonging to the group 7 are:
 22 carburetor
 37 carburetor
 58 carburetor

 rules belonging to the group 8 are:
 39 exhaust

 rules belonging to the group 9 are:
 41 drive-train

 rules belonging to the group 10 are:
 42 fuel
 60 ammeter
 55 fuel
 56 fuel,carburetor,cylinder
 59 ammeter

 rules belonging to the group 11 are:
 44 distributor+s
 46 distributor+s
 54 fuel+s

 rules belonging to the group 12 are:
 47 ammeter
 48 distributor,ammeter
 52 distributor,ammeter
 50 distributor,ammeter

 ------------------------------------------------------------------------
```

```
CAR : DATA_FLOW : V : 12 groups

**** DISTANCE METRIC => DATA_FLOW ****

The global stopping criteria is 12 groups:

rules belonging to the group 1 are:
1 battery,starter,engine
3 battery,engine,cylinder
4 battery,engine,cylinder
43 battery,engine
5 cylinder
2 battery,starter,engine
32 battery
31 starter,battery,engine
34 engine
30 starter,battery,engine
33 starter
40 engine

rules belonging to the group 2 are:
6 cylinder
10 sp_plug
11 sp_plug
19 sp_plug
12 sp_plug
13 sp_plug
14 sp_plug
15 sp_plug
16 sp_plug
17 sp_plug

rules belonging to the group 3 are:
7 cylinder,sp_plug,distributor
8 distributor
9 cylinder
21 distributor
23 cylinder
38 cylinder
44 distributor+s
45 distributor
48 distributor,ammeter
49 distributor
50 distributor,ammeter
51 distributor
52 distributor,ammeter
53 distributor
```

```
24 engine,cylinder,sp_plug,distributor
25 sp_plug
26 distributor
27 cylinder

rules belonging to the group - are:
18 engine,cylinder,sp_plug,ignition,carburetor,distributor
20 ignition
22 carburetor

rules belonging to the group 5 are:
28 starter+l

rules belonging to the group 6 are:
29 starter+l

rules belonging to the group 7 are:
35 battery,engine,carburetor,distributor,exhaust,drive-train
36 distributor
37 carburetor

rules belonging to the group 8 are:
39 exhaust

rules belonging to the group 9 are:
41 drive-train

rules belonging to the group 10 are:
42 fuel

rules belonging to the group 11 are:
46 distributor+s
47 ammeter

rules belonging to the group 12 are:
54 fuel+s
55 fuel
59 ammeter
60 ammeter
56 fuel,carburetor,cylinder
57 cylinder
58 carburetor
```

Appendix C - MMU-FDIR RULE BASE

```
;; The first 10 rules perform input-output and control firing of rules. Hence
;; they do not play a role in the grouping and have not been numbered.

(defrule very-last-rule
(declare (salience -100))
(fact-name ?name)
=>
(printout crlf "test case is complete, return any character to continue" crlf)
(assert (last-entry =(read)))
(system "cls")
(undeffacts (?name))
(reset)
(system "cls")
)

;;Gathering state information

(defrule next-to-last
(declare (salience -50))
(side a ?state-a)
(side b ?state-b)
=>
(printout crlf "side A is "?state-a crlf)
(printout "side B is "?state-b crlf)
)

;;Print

(defrule print-cea-test
(declare (salience 90))
(print)
=>
(printout crlf "testing cea" crlf)
)

(defrule print-tank-test
(declare (salience -5))
(print)
(not (failure ?))
=>
(printout crlf "testing tank pressure and thrusters"crlf)
)

;;simulation print

(defrule print-zero
```

```
(declare (salience 100))
(zero)
=>
(system "cls")
(printout crlf "************************************************************"crlf)
(printout "Command: "crlf)
(printout "Translation in the pos x direction "crlf crlf)
(printout "Expected failure: "crlf)
(printout "Cea failure - a signal from the valve drive amp on side A "crlf)
(printout "              was not sent to thrusters."crlf crlf)
(printout "Initial state: "crlf)
(printout "Side A is on, side B is on"crlf crlf)
(printout "Expected final state:"crlf)
(printout "Side A off, side B on" crlf)
(printout crlf "************************************************************"crlf)
(printout crlf crlf "return any character to continue" crlf)
(assert (print-zero-a =(read)))
(system "cls")
)

(defrule print-one
(declare (salience 100))
(one)
=>
(system "cls")
(printout crlf "************************************************************"crlf)
(printout "Command: "crlf)
(printout "Translation in the pos y direction "crlf crlf)
(printout "Expected failure: "crlf)
(printout "Cea failure - uncommanded signal from the valve drive amp on "crlf)
(printout "              side B was sent to thrusters."crlf crlf)
(printout "Initial state: "crlf)
(printout "Side A is on, side B is on"crlf crlf)
(printout "Expected final state:"crlf)
(printout "Side A on, side B off" crlf)
(printout crlf "************************************************************"crlf)
(printout crlf crlf "return any character to continue" crlf)
(assert (print-one-a =(read)))
(system "cls")
)

(defrule print-two
(declare (salience 100))
(two)
=>
(system "cls")
(printout crlf "************************************************************"crlf)
(printout "Command: "crlf)
```

```
(printout "Translation in the neg y direction "crlf crlf)
(printout "Expected failure: "crlf)
(printout "Thruster failure - tank pressure on side A is high, a thruster "crlf)
(printout "                   on side A has failed to respond."crlf crlf)
(printout "Initial state: "crlf)
(printout "Side A is on, side B is on" crlf crlf)
(printout "Expected final state:"crlf)
(printout "Side A off, side B on" crlf)
(printout crlf "**********************************************************"crlf)
(printout crlf crlf "return any character to continue" crlf)
(assert (print-two-a =(read)))
(system "cls")
)

(defrule print-three
(declare (salience 100))
(three)
=>
(system "cls")
(printout crlf "**********************************************************"crlf)
(printout "Command: " crlf)
(printout "Rotation in the pos roll direction " crlf crlf)
(printout "Expected failure: " crlf)
(printout "Thruster failure - Tank pressure on side B is low.  "crlf)
(printout "                   Possible uncommanded acceleration" crlf)
(printout "                   or fuel leak has occurred." crlf crlf)
(printout "Initial state: " crlf)
(printout "Side A is on, side B is on" crlf crlf)
(printout "Expected final state:" crlf)
(printout "Side A on, side B off" crlf)
(printout crlf "**********************************************************"crlf)
(printout crlf crlf "return any character to continue" crlf)
(assert (print-three-a =(read)))
(system "cls")
)

(defrule print-four
(declare (salience 100))
(four)
=>
(system "cls")
(printout crlf "**********************************************************"crlf)
(printout "Command: "crlf)
(printout "Translation in the pos z direction "crlf crlf)
(printout "Expected failure: "crlf)
(printout "Thruster failure - Tank pressure is low during xfeed. After  " crlf)
(printout "                   isolation, tank pressure on side B is low." crlf)
(printout "                   uncommanded acceleration" crlf)
```

```
(printout "                    or fuel leak has occurred." crlf crlf)
(printout "Initial state: "crlf)
(printout "Xfeed A is open, xfeed B is on"crlf)
(printout "Side A is on, side B is on"crlf crlf)
(printout "Expected final state:"crlf)
(printout "Side A on, side B off" crlf)
(printout crlf "*************************************************************"crlf)
(printout crlf crlf "return any character to continue" crlf)
(assert (print-four-a =(read)))
(system "cls")
)

(defrule print-five
(declare (salience 100))
(five)
=>
(system "cls")
(printout crlf "*************************************************************"crlf)
(printout "Command: "crlf)
(printout "No command is given"crlf crlf)
(printout "Expected failure: "crlf)
(printout "Cea failure - an uncommanded neg pitch has occurred  "crlf)
(printout "              attitude hold fails to correct."crlf crlf)
(printout "              Both cea-a and cea-b will fail." crlf crlf)
(printout "Initial state: "crlf)
(printout "Side A is on, side B is on"crlf)
(printout "Attitude hold is on, gyro is on" crlf crlf)
(printout "Expected final state:"crlf)
(printout "Side A off, side B off" crlf)
(printout crlf "*************************************************************"crlf)
(printout crlf crlf "return any character to continue" crlf)
(assert (print-five-a =(read)))
(system "cls")
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;improper CEA behavior
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;logic for (no aah) or (no gyro movement)and(aah on) - prime mode
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule cea-test-input-null ;rule_no 1
(or (aah off) (and (gyro on)(gyro_movement none none)))
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(vda ?side ?thrust on)
=>
(assert (failure cea))
(printout crlf "failure - vda signal was sent to " crlf)
```

```
(printout "thrusters without intended command "crlf)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;logic for x, pitch, yaw
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;pos x input
(defrule cea-a-test-input-posx-null-null-1 ;rule_no 2
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x pos y none z none)
(or
(vda a f2 off)
(vda a f3 off)
(vda a ?n&~f1&~f2&~f3&~f4 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during translational command ")
(printout "in the pos x direction" crlf)
)

(defrule cea-b-test-input-posx-null-null-1 ;rule_no 3
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x pos y none z none)
(or
        (vda b f1 off)
(vda b f4 off)
(vda b ?n&~f1&~f2&~f3&~f4 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during translational command ")
(printout "in the pos x direction" crlf)
)

;neg x input
(defrule cea-b-test-input-negx-null-null-2 ;rule_no 4
(or (aah off) (and (gyro on)(gyro_movement none none)))
```

```
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x neg y none z none)
(or
  (vda b b2 off)
(vda b b3 off)
(vda b ?n&~b1&~b2&~b3&~b4 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during translational command ")
(printout "in the neg x direction" crlf)
)

(defrule cea-a-test-input-negx-null-null-2 ;rule_no 5
(or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x neg y none z none)
(or
(vda a b1 off)
(vda a b4 off)
(vda a ?n&~b1&~b2&~b3&~b4 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during translational command ")
(printout "in the neg x direction" crlf)
)

;pos pitch input
(defrule cea-a-test-input-null-pos-null-3 ;rule_no 6
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch pos yaw none)
(thc x none y none z none)
(or
(vda a b1 off)
(vda a f3 off)
(vda a ?n&~b1&~f3 on)
)
=>
(assert (failure cea))
```

```
(assert (suspect a))
(printout crlf "failure -during rotational command ")
(printout "in the pos pitch direction" crlf)
)

(defrule cea-b-test-input-null-pos-null-3 ;rule_no 7
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch pos yaw none)
(thc x none y none z none)
(vda b ?m on)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during rotational command ")
(printout "in the pos pitch direction" crlf)
)

;neg pitch
(defrule cea-b-test-input-null-neg-null-4 ;rule_no 8
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch neg yaw none)
(thc x none y none z none)
(or
(vda b f1 off)
(vda b b3 off)
(vda b ?n&~b3&~f1 on)
  )
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during rotational command ")
(printout "in the neg pitch direction" crlf)
)

(defrule cea-a-test-input-null-neg-null-4 ;rule_no 9
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch neg yaw none)
(thc x none y none z none)
(vda a ?m on)
=>
(assert (failure cea))
(assert (suspect a))
```

```
(printout crlf "failure -during rotational command ")
(printout "in the neg pitch direction" crlf)
)

;pos yaw
(defrule cea-a-test-input-null-null-pos-5 ;rule_no 10
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw pos)
(thc x none y none z none)
(or
(vda a f2 off)
(vda a b1 off)
(vda a ?n&~b1&~f2 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during rotational command ")
(printout "in the pos yaw direction" crlf)
)

(defrule cea-b-test-input-null-null-pos-5 ;rule_no 11
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw pos)
(thc x none y none z none)
(vda b ?m on)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during rotational command ")
(printout "in the pos yaw direction" crlf)
)

;neg yaw
(defrule cea-b-test-input-null-null-pos-6 ;rule_no 12
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw neg)
(thc x none y none z none)
(or
(vda b f1 off)
(vda b b2 off)
(vda b ?n&~b2&~f1 on)
```

```
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during rotational command ")
(printout "in the neg yaw direction" crlf)
)

(defrule cea-a-test-input-null-null-pos-6 ;rule_no 13
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw neg)
(thc x none y none z none)
(vda a ?m on)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during rotational command ")
(printout "in the neg yaw direction" crlf)
)
;
;
;;;;;;;;;;;;;;;;;;;;;;;;
;logic for y, roll, yaw
;;;;;;;;;;;;;;;;;;;;;;;;

;pos y,
(defrule cea-a-test-input-posy-null-null-7 ;rule_no 14
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y pos z none)
(or
(vda a r2 off)
(vda a r4 off)
(vda a ?n&~r2&~r4 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during translational command ")
(printout "in the pos y direction" crlf)
)

(defrule cea-b-test-input-posy-null-null-7 ;rule_no 15
        (or (aah off) (and (gyro on)(gyro_movement none none)))
```

```
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y pos z none)
(or
(vda b r2 off)
(vda b r4 off)
(vda b ?n&~r2&~r4 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during translational command ")
(printout "in the pos y direction" crlf)
)

;neg y
(defrule cea-a-test-input-neg-null-null-8 ;rule_no 16
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y neg z none)
(or
(vda a l1 off)
(vda a l3 off)
(vda a ?n&~l1&~l3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during translational command ")
(printout "in the neg y direction" crlf)
)

(defrule cea-b-test-input-neg-null-null-8 ;rule_no 17
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y neg z none)
(or
(vda b l1 off)
(vda b l3 off)
(vda b ?n&~l1&~l3 on)
)
=>
(assert (failure cea))
```

```
(assert (suspect b))
(printout crlf "failure -during translational command ")
(printout "in the neg y direction" crlf)
)

;pos roll
(defrule cea-a-test-input-null-pos-null-9 ;rule_no 18
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll pos pitch none yaw none)
(thc x none y none z none)
(or
(vda a r2 off)
(vda a l3 off)
(vda a ?n&~r2&~l3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during rotational command ")
(printout "in the pos roll direction" crlf)
)

(defrule cea-b-test-input-null-pos-null-9 ;rule_no 19
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll pos pitch none yaw none)
(thc x none y none z none)
(vda b ?m on)

=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during rotational command ")
(printout "in the pos roll direction" crlf)
)

;neg roll
(defrule cea-a-test-input-null-neg-null-10 ;rule_no 20
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll neg pitch none yaw none)
(thc x none y none z none)
(vda a ?m on)
```

```
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during rotational command ")
(printout "in the neg roll direction" crlf)
)

(defrule cea-b-test-input-null-neg-null-10 ;rule_no 21
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll neg pitch none yaw none)
(thc x none y none z none)
(or
(vda b r4 off)
(vda b l1 off)
(vda b ?n&~r4&~l1 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during rotational command ")
(printout "in the neg roll direction" crlf)
)
;
;;;;;;;;;;;;;;;;;;;;;;;;;;
;logic for z, roll, pitch
;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;pos z,
(defrule cea-a-test-input-posz-null-null-11 ;rule_no 22
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z pos)
(or
(vda a d1 off)
(vda a d2 off)
(vda a ?n&~d1&~d2 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during translational command ")
(printout "in the pos z direction" crlf)
)
```

```
(defrule cea-b-test-input-posz-null-null-11 ;rule_no 23
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z pos)
(or
(vda b d1 off)
(vda b d2 off)
(vda b ?n&~d1&~d2 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure -during translational command ")
(printout "in the pos z direction" crlf)
)

;neg z
(defrule cea-a-test-input-neg-null-null-12 ;rule_no 24
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z neg)
(or
(vda a u3 off)
(vda a u4 off)
(vda a ?n&~u3&~u4 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "failure -during translational command ")
(printout "in the neg z direction" crlf)
)

(defrule cea-b-test-input-neg-null-null-12 ;rule_no 25
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z neg)
(or
(vda b u3 off)
(vda b u4 off)
(vda b ?n&~u3&~u4 on)
)
```

```
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "failure during translational command ")
(printout "in the neg z direction" crlf)
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;gyro movement rules - (axis direction) - prime mode
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;negative pitch gyro indications
(defrule cea-a-gyro-input-pitch-neg-1 ;rule_no 26
        (aah on)  (gyro on)
        (gyro_movement pitch neg)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda a b1 off)
(vda a f3 off)
(vda a ?n&~b1&~f3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "aah failed to correct neg pitch")
)

(defrule cea-b-gyro-input-pitch-neg-1 ;rule_no 27
        (aah on)  (gyro on)
        (gyro_movement pitch neg)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(vda b ?m on)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "aah failed to correct neg pitch")
)

;pos pitch gyro indications
(defrule cea-b-gyro-input-pitch-pos-2 ;rule_no 28
        (aah on)  (gyro on)
```

```
        (gyro_movement pitch pos)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda b f1 off)
(vda b b3 off)
(vda b ?n&~b3&~f1 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "aah failed to correct pos pitch")
)

(defrule cea-a-gyro-input-pitch-pos-2 ;rule_no 29
        (aah on)  (gyro on)
        (gyro_movement pitch pos)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(vda a ?m on)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "aah failed to correct pos pitch")
)

;neg yaw gyro indication
(defrule cea-a-gyro-input-yaw-neg-3 ;rule_no 30
        (aah on) (gyro on)
        (gyro_movement yaw neg)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda a f2 off)
(vda a b1 off)
(vda a ?n&~b1&~f2 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "aah failed to correct neg yaw")
)
```

```
(defrule cea-b-gyro-input-yaw-neg-3 ;rule_no 31
        (aah on) (gyro on)
        (gyro_movement yaw neg)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(vda b ?m on)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "aah failed to correct neg yaw")
)

;pos yaw gyro indication
(defrule cea-b-gyro-input-yaw-pos-4 ;rule_no 32
        (aah on) (gyro on)
        (gyro_movement yaw pos)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda b f1 off)
(vda b b2 off)
(vda b ?n&~b2&~f1 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "aah failed to correct pos yaw")
)

(defrule cea-a-gyro-input-yaw-pos-4 ;rule_no 33
        (aah on) (gyro on)
        (gyro_movement yaw pos)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(vda a ?m on)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "aah failed to correct pos yaw")
)
```

```
;neg roll gyro indication
(defrule cea-a-gyro-input-roll-neg-5 ;rule_no 34
        (aah on) (gyro on)
        (gyro_movement roll neg)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda a r2 off)
(vda a l3 off)
(vda a ?n&~r2&~l3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "aah failed to correct neg roll")
)

(defrule cea-b-gyro-input-roll-neg-5 ;rule_no 35
        (aah on) (gyro on)
        (gyro_movement roll neg)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(vda b ?m on)

=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "aah failed to correct neg roll")
)
;


;pos roll gyro input
(defrule cea-a-gyro-input-roll-pos-6 ;rule_no 36
        (aah off) (gyro on)
        (gyro_movement roll pos)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(vda a ?m on)

=>
(assert (failure cea))
```

```
(assert (suspect a))
(printout crlf "aah failed to correct pos roll")
)

(defrule cea-b-gyro-input-roll-pos-6 ;rule_no 37
        (aah off) (gyro on)
        (gyro_movement roll pos)
(side a on)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda b r4 off)
(vda b l1 off)
(vda b ?n&~r4&~l1 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "aah failed to correct pos roll")
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; back up mode logic for side a - no gyro
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;
;;logic for x, pitch, yaw
;;;;;;;;;;;;;;;;;;;;;;;;;;

;
;pos x
(defrule cea-test-input-pos-null-null-side-a-1 ;rule_no 38
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x pos y none z none)
(or
(vda a f2 off)
(vda a f3 off)
(vda a ?n&~f2&~f3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)
```

```
;neg x
(defrule cea-test-input-neg-null-null-side-a-2 ;rule_no 39
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x neg y none z none)
(or
(vda a b1 off)
(vda a b4 off)
(vda a ?n&~b1&~b4 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;pos pitch
(defrule cea-test-input-null-pos-null-side-a-3 ;rule_no 40
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch pos yaw none)
(thc x none y none z none)
(or
(vda a b1 off)
(vda a f3 off)
(vda a ?n&~b1&~f3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;neg pitch
(defrule cea-test-input-null-neg-null-side-a-4 ;rule_no 41
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch neg yaw none)
(thc x none y none z none)
(or
```

```
(vda a f2 off)
(vda a b4 off)
(vda a ?n&~b4&~f2 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;pos yaw
(defrule cea-test-input-null-null-pos-side-a-5 ;rule_no 42
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw pos)
(thc x none y none z none)
(or
(vda a b1 off)
(vda a f2 off)
(vda a ?n&~b1&~f2 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;neg yaw
(defrule cea-test-input-null-null-neg-side-a-6 ;rule_no 43
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw neg)
(thc x none y none z none)
(or
(vda a b4 off)
(vda a f3 off)
(vda a ?n&~b4&~f3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)
;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;
;;logic for y, z, roll
;;;;;;;;;;;;;;;;;;;;;;;;

;pos y
(defrule cea-test-input-pos-null-null-side-a-7 ;rule_no 44
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x none y pos z none)
(or
(vda a r2 off)
(vda a r4 off)
(vda a ?n&~r2&~r4 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;neg y
(defrule cea-test-input-neg-null-null-side-a-8 ;rule_no 45
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x none y neg z none)
(or
(vda a l1 off)
(vda a l3 off)
(vda a ?n&~l1&~l3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;pos z
(defrule cea-test-input-null-pos-null-side-a-9 ;rule_no 46
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
```

```
(rhc roll none pitch none yaw none)
(thc x none y none z pos)
(or
(vda a d1 off)
(vda a d2 off)
(vda a ?n&~d1&~d2 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;neg z
(defrule cea-test-input-null-neg-null-side-a-10 ;rule_no 47
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x none y none z neg)
(or
(vda a u3 off)
(vda a u4 off)
(vda a ?n&~u3&~u4 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;pos roll
(defrule cea-test-input-null-null-pos-side-a-11 ;rule_no 48
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll pos pitch none yaw none)
(thc x none y none z none)
(or
(vda a r2 off)
(vda a l3 off)
(vda a ?n&~r2&~l3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
```

```
(printout crlf "cea failure on side a")
)

;neg roll
(defrule cea-test-input-null-null-neg-side-a-12 ;rule_no 49
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll neg pitch none yaw none)
(thc x none y none z none)
(or
(vda a r4 off)
(vda a l1 off)
(vda a ?n&~r4&~l1 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; backup logic for side b - no gyro
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;logic for x, pitch, yaw
;;;;;;;;;;;;;;;;;;;;;;;;;;;

;
;pos x
(defrule cea-test-input-pos-null-null-side-b-1 ;rule_no 50
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x pos y none z none)
(or
(vda b f1 off)
(vda b f4 off)
(vda b ?n&~f1&~f4 on)
)
=>
(assert (failure cea))
(assert (suspect b))
```

```
(printout crlf "cea failure on side b")
)

;neg x
(defrule cea-test-input-neg-null-null-side-b-2 ;rule_no 51
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x neg y none z none)
(or
(vda b b2 off)
(vda b b3 off)
(vda b ?n&~b2&~b3 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;pos pitch
(defrule cea-test-input-null-pos-null-side-b-3 ;rule_no 52
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch pos yaw none)
(thc x none y none z none)
(or
(vda b f4 off)
(vda b b2 off)
(vda b ?n&~b2&~f4 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;neg pitch
(defrule cea-test-input-null-neg-null-side-b-4 ;rule_no 53
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch neg yaw none)
```

```
(thc x none y none z none)
(or
(vda b f1 off)
(vda b b3 off)
(vda b ?n&~f1&~b3 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;pos yaw
(defrule cea-test-input-null-null-pos-side-b-5 ;rule_no 54
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw pos)
(thc x none y none z none)
(or
(vda b f4 off)
(vda b b3 off)
(vda b ?n&~b3&~f4 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;neg yaw
(defrule cea-test-input-null-null-neg-side-b-6 ;rule_no 55
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw neg)
(thc x none y none z none)
(or
(vda b f1 off)
(vda b b2 off)
(vda b ?n&~f1&~b2 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
```

```
)
;
;;;;;;;;;;;;;;;;;;;;;;;;
;;logic for y, z, roll
;;;;;;;;;;;;;;;;;;;;;;;;

;pos y
(defrule cea-test-input-pos-null-null-side-b-7 ;rule_no 56
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y pos z none)
(or
(vda b r2 off)
(vda b r4 off)
(vda b ?n&~r2&~r4 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;neg y
(defrule cea-test-input-neg-null-null-side-b-8 ;rule_no 57
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y neg z none)
(or
(vda b l1 off)
(vda b l3 off)
(vda b ?n&~l1&~l3 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;pos z
(defrule cea-test-input-null-pos-null-side-b-9 ;rule_no 58
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
```

```
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z pos)
(or
(vda b d1 off)
(vda b d2 off)
(vda b ?n&~d1&~d2 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;neg z
(defrule cea-test-input-null-neg-null-side-b-10 ;rule_no 59
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z neg)
(or
(vda b u3 off)
(vda b u4 off)
(vda b ?n&~u3&~u4 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;pos roll
(defrule cea-test-input-null-null-pos-side-b-11 ;rule_no 60
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll pos pitch none yaw none)
(thc x none y none z none)
(or
(vda b r2 off)
(vda b l3 off)
(vda b ?n&~r2&~l3 on)
)
=>
```

```
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;neg roll
(defrule cea-test-input-null-null-neg-side-b-12 ;rule_no 61
        (or (aah off) (and (gyro on)(gyro_movement none none)))
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll neg pitch none yaw none)
(thc x none y none z none)
(or
(vda b r4 off)
(vda b l1 off)
(vda b ?n&~r4&~l1 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;gyro movement rules - (axis direction) - backup mode
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;negative pitch gyro indications
(defrule gyro-input-pitch-neg-backup-b-1 ;rule_no 62
        (aah on)  (gyro on)
        (gyro_movement pitch neg)
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda b f4 off)
(vda b b2 off)
(vda b ?n&~f4&~b2 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)
```

```
(defrule gyro-input-pitch-neg-backup-a-1 ;rule_no 63
        (aah on)  (gyro on)
        (gyro_movement pitch neg)
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda a b1 off)
(vda a f3 off)
(vda a ?n&~b1&~f3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;pos pitch gyro indications
(defrule gyro-input-pitch-pos-backup-a-2 ;rule_no 64
        (aah on)  (gyro on)
        (gyro_movement pitch pos)
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda a f2 off)
(vda a b4 off)
(vda a ?n&~f2&~b4 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

(defrule gyro-input-pitch-pos-backup-b-2 ;rule_no 65
        (aah on)  (gyro on)
        (gyro_movement pitch pos)
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
```

```
(or
(vda b f1 off)
(vda b b3 off)
(vda b ?n&~f1&~b3 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

;neg yaw gyro indication
(defrule gyro-input-yaw-neg-backup-b-3 ;rule_no 66
        (aah on) (gyro on)
        (gyro_movement yaw neg)
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda b f4 off)
(vda b b3 off)
(vda b ?n&~f4&~b3 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

(defrule gyro-input-yaw-neg-backup-a-3 ;rule_no 67
        (aah on) (gyro on)
        (gyro_movement yaw neg)
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda a f2 off)
(vda a b1 off)
(vda a ?n&~f2&~b1 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
```

```
 )

 ;pos yaw gyro indication
 (defrule gyro-input-yaw-pos-backup-a-4 ;rule_no 68
         (aah on) (gyro on)
         (gyro_movement yaw pos)
 (not (checking thrusters))
 (side a on)
 (side b off)
 (rhc roll none pitch none yaw none)
 (thc x none y none z none)
 (or
 (vda a f3 off)
 (vda a b4 off)
 (vda a ?n&~b4&~f3 on)
 )
 =>
 (assert (failure cea))
 (assert (suspect a))
 (printout crlf "cea failure on side a")
 )

 (defrule gyro-input-yaw-pos-backup-b-4 ;rule_no 69
         (aah on) (gyro on)
         (gyro_movement yaw pos)
 (not (checking thrusters))
 (side a off)
 (side b on)
 (rhc roll none pitch noné yaw none)
 (thc x none y none z none)
 (or
 (vda b f1 off)
 (vda b b2 off)
 (vda b ?n&~b2&~f1 on)
 )
 =>
 (assert (failure cea))
 (assert (suspect b))
 (printout crlf "cea failure on side b")
 )

 ;neg roll gyro indication
 (defrule gyro-input-roll-neg-backup-b-5 ;rule_no 70
         (aah on) (gyro on)
         (gyro_movement roll neg)
 (not (checking thrusters))
 (side a off)
 (side b on)
```

```
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda b r2 off)
(vda b l3 off)
(vda b ?n&~r2&~l3 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)

(defrule gyro-input-roll-neg-backup-a-5 ;rule_no 71
        (aah on) (gyro on)
        (gyro_movement roll neg)
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda a r2 off)
(vda a l3 off)
(vda a ?n&~r2&~l3 on)
)
=>
(assert (failure cea))
(assert (suspect a))
(printout crlf "cea failure on side a")
)

;pos roll gyro input
(defrule gyro-input-roll-pos-backup-a-6 ;rule_no 72
        (aah off) (gyro on)
        (gyro_movement roll pos)
(not (checking thrusters))
(side a on)
(side b off)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda a r4 off)
(vda a l1 off)
(vda a ?n&~r4&~l1 on)
)
=>
(assert (failure cea))
```

```
(assert (suspect a))
(printout crlf "cea failure on side a")
)

(defrule gyro-input-roll-pos-backup-b-6 ;rule_no 73
        (aah off) (gyro on)
        (gyro_movement roll pos)
(not (checking thrusters))
(side a off)
(side b on)
(rhc roll none pitch none yaw none)
(thc x none y none z none)
(or
(vda b r4 off)
(vda b l1 off)
(vda b ?n&~r4&~l1 on)
)
=>
(assert (failure cea))
(assert (suspect b))
(printout crlf "cea failure on side b")
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; cea-failure recovery
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;cea a rules
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule test-failure-cea-suspect-a ;rule_no 74
?a <- (failure cea)
(suspect a)
(side a on)
?b <- (side b on)
=>
(retract ?a ?b)
(assert (side b off))
(printout crlf "suspected cea failure"crlf)
(printout crlf "turning side-b off "crlf)
(printout crlf "recalling command inputs"crlf)
(printout crlf "testing cea-a"crlf)
)

(defrule test-failure-cea-a-good ;rule_no 75
(not (failure cea))
?x <- (suspect a)
?a <- (side b off)
?b <- (side a on)
=>
```

```
 (retract ?a ?b ?x)
 (assert (side b on))
 (assert (side a off))
 (assert (cea-a-good))
 (printout crlf "cea-a was suspected and tested but responds correctly")
 (printout crlf "turning side-a off, side-b on, testing cea-b"crlf)
 )

 (defrule test-failure-cea-a-bad ;rule_no 76
 ?a <- (failure cea)
 (suspect a)
 ?b <- (side b off)
 ?c <- (side a on)
 (not (failure cea-b))
 =>
 (retract ?a ?b ?c)
 (assert (side b on))
 (assert (side a off))
 (assert (failure cea-a))
 (printout crlf "setting side-a off and side-b on , testing cea-b"crlf)
 )



 (defrule test-a-cea-side-b-good ;rule_no 77
 (not (failure cea))
 (side b on)
 (side a off)
 (cea-a-good)
 =>
 (assert (failure cea-coupled))
 (printout crlf "coupled cea failure" crlf)
 (printout crlf "side-a is off, side-b is on" crlf)
 )

 (defrule test-a-cea-side-b-bad ;rule_no 78
 (failure cea)
 ?x <- (side b on)
 ?y <- (side a off)
 (not (suspect b))
 =>
 (retract ?x ?y)
 (assert (side b off))
 (assert (side a on))
 (printout crlf "cea-b failed, side-a is on")
 )

 (defrule test-a-cea-side-a-and-b ;rule_no 79
```

```
(failure cea)
(failure cea-a)
?x <- (side b on)
(side a off)
=>
(retract ?x)
(assert (failure cea-a-b))
(assert (side b off))
(printout crlf "cea-a and cea-b have both failed, call for help" crlf)
)

(defrule print-failure-cea-a ;rule_no 80
(declare (salience -8))
(side a off)
(side b on)
(not (failure cea))
(failure cea-a)
=>
(printout crlf "cea-b responds correctly"crlf)
)
;;;;;;;;;;;;;;;;;;;;
;;;;cea-b rules
;;;;;;;;;;;;;;;;;;;;
(defrule test-failure-cea-suspect-b ;rule_no 81
?a <- (failure cea)
(suspect b)
?b <- (side a on)
(side b on)
=>
(retract ?a ?b)
(assert (side a off))
(printout crlf "suspected cea failure" crlf)
(printout crlf "turning side-a off " crlf)
(printout crlf "recalling command inputs" crlf)
(printout crlf "testing cea-b" crlf)
)

(defrule test-failure-cea-b-good ;rule_no 82
(not (failure cea))
?x <- (suspect b)
?a <- (side a off)
?b <- (side b on)
=>
(retract ?a ?b ?x)
(assert (side a on))
(assert (side b off))
(assert (cea-b-good))
(printout crlf "cea-b was suspected and tested but responds correctly")
```

```
(printout crlf "turning side-b off, side-a on, testing cea-a"crlf)
)

(defrule test-failure-cea-b-bad ;rule_no 83
?a <- (failure cea)
(suspect b)
?b <- (side a off)
?c <- (side b on)
(not (failure cea-a))
=>
(retract ?a ?b ?c)
(assert (side a on))
(assert (side b off))
(assert (failure cea-b))
(printout crlf "setting side-b off and side-a on , testing cea-a"crlf)
)

(defrule test-b-cea-side-a-good ;rule_no 84
(not (failure cea))
(side a on)
(side b off)
(cea-b-good)
=>
(assert (failure cea-coupled))
(printout crlf "coupled cea failure" crlf)
(printout crlf "side-b is off, side-a is on" crlf)
)

(defrule test-b-cea-side-a-bad ;rule_no 85
(failure cea)
?x <- (side a on)
?y <- (side b off)
(not (suspect a))
=>
(retract ?x ?y)
(assert (side a off))
(assert (side b on))
(printout crlf "cea-a failed, side-b is on")
)

(defrule test-b-cea-side-a-and-b ;rule_no 86
(failure cea)
(failure cea-b)
?x <- (side a on)
(side b off)
=>
(retract ?x)
(assert (failure cea-a-b))
```

```
(assert (side a off))
(printout crlf "cea-a and cea-b have both failed, call for help" crlf)
)


(defrule print-failure-cea-b ;rule_no 87
(declare (salience -8))
(side b off)
(side a on)
(not (failure cea))
(failure cea-b)
=>
(printout crlf "side a responds correctly"crlf)
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; thruster-failure
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; test for thruster firing failure
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule no-xfeed-fuel-calculation-side-a ;rule_no 88
(declare (salience 10))
(xfeed-a closed)
(xfeed-b closed)
(vda a ?n on)
(not (read-it a ?n))
(not (failure ?))
?x <- (fuel-used-a ?fuel-a)
=>
(retract ?x)
(assert (read-it a ?n))
(assert (fuel-used-a =(+ ?fuel-a 1)))
)

(defrule no-xfeed-fuel-calculation-side-b ;rule_no 89
(declare (salience 10))
(xfeed-a closed)
(xfeed-b closed)
(vda b ?n on)
(not (read-it b ?n))
(not (failure ?))
?x <- (fuel-used-b ?fuel-b)
=>
(retract ?x)
```

```
(assert (read-it b ?n))
(assert (fuel-used-b =(+ ?fuel-b 1)))
)

(defrule no-xfeed-fuel-reading-test-side-a-grt ;rule_no 90
(declare (salience -10))
(xfeed-a closed)
(xfeed-b closed)
(not (failure ?))
(fuel-used-a ?fuel-a)
(tank-pressure-was a ?p-old)
(tank-pressure-current a ?p-new)
(test (< (- ?p-old ?fuel-a) ?p-new))
?x <- (side a on)
(side b on)
=>
(assert (failure thruster-a))
(printout crlf "pressure in tank a is high, a thruster has not responded"crlf)
(printout crlf "side a failed while executing thruster commands" crlf)
(assert (side a off))
(retract ?x)
(assert (checking thrusters))
)

(defrule no-xfeed-fuel-reading-test-side-a-lss ;rule_no 91
(declare (salience -10))
(xfeed-a closed)
(xfeed-b closed)
(not (failure ?))
(fuel-used-a ?fuel-a)
(tank-pressure-was a ?p-old)
(tank-pressure-current a ?p-new)
(test (> (- ?p-old ?fuel-a) ?p-new))
?x <- (side a on)
(side b on)
=>
(assert (failure thruster-a))
(printout crlf "pressure in tank a is low, " crlf)
(printout "possible uncommanded acceleration or fuel leak" crlf crlf)
(printout "side a failed while executing thruster commands" crlf)
(assert (side a off))
(retract ?x)
(assert (checking thrusters))
)

(defrule no-xfeed-fuel-reading-test-side-b-grt ;rule_no 92
(declare (salience -10))
(xfeed-a closed)
```

```
(xfeed-b closed)
(not (failure ?))
(fuel-used-b ?fuel-b)
(tank-pressure-was b ?p-old)
(tank-pressure-current b ?p-new)
(test (< (- ?p-old ?fuel-b) ?p-new))
(side a on)
?x <- (side b on)
=>
(assert (failure thruster-b))
(printout crlf "pressure in tank b is high, a thruster has not responded"crlf)
(printout crlf "side b failed while executing thruster commands" crlf)
(assert (side b off))
(retract ?x)
(assert (checking thrusters))
)

(defrule no-xfeed-fuel-reading-test-side-b-lss ;rule_no 93
(declare (salience -10))
(xfeed-a closed)
(xfeed-b closed)
(not (failure ?))
(fuel-used-b ?fuel-b)
(tank-pressure-was b ?p-old)
(tank-pressure-current b ?p-new)
(test (> (- ?p-old ?fuel-b) ?p-new))
(side a on)
?x <- (side b on)
=>
(assert (failure thruster-b))
(printout crlf "pressure in tank b is low, " crlf)
(printout "possible uncommanded acceleration or fuel leak" crlf crlf)
(printout crlf "side b failed while executing thruster commands" crlf)
(assert (side b off))
(retract ?x)
(assert (checking thrusters))
)

(defrule xfeed-fuel-reading-test-general ;rule_no 94
(declare (salience -10))
?x <- (xfeed-a open)
?y <- (xfeed-b open)
(fuel-used-a ?fuel-a)
(fuel-used-b ?fuel-b)
(tank-pressure-was ab ?p-old)
(tank-pressure-current ab ?p-new)
(test (!= (- ?p-old (+ ?fuel-a ?fuel-b)) ?p-new))
(side b on)
```

```
(side a on)
=>
(retract ?x ?y)
(assert (xfeed-a closed))
(assert (xfeed-b closed))
(assert (failure-thrusters-with-xfeed))
(printout crlf "failure occurred while executing thruster commands")
(printout crlf crlf "xfeed is open, testing sides after closing xfeed")
(printout crlf crlf)
(assert (checking thrusters))
)

--------------------------------------------------------------------------------
Key concepts carried by the rules
--------------------------------------------------------------------------------

1 improper cea behavior
2 logic for x, pitch and yaw
5 logic for x, pitch and yaw
7 logic for x, pitch and yaw
8 logic for x, pitch and yaw
9 logic for x, pitch and yaw
10 logic for x, pitch and yaw
11 logic for x, pitch and yaw
12 logic for x, pitch and yaw
13 logic for x, pitch and yaw
14 logic for x, pitch and yaw
15  logic for y, roll and yaw
16  logic for y, roll and yaw
17  logic for y, roll and yaw
18  logic for y, roll and yaw
19  logic for y, roll and yaw
20  logic for y, roll and yaw
21  logic for y, roll and yaw
22  logic for z, roll and pitch
23  logic for z, roll and pitch
24  logic for z, roll and pitch
25  logic for z, roll and pitch
26 gyro movement rules- axis direction- prime mode
27 gyro movement rules- axis direction- prime mode
28 gyro movement rules- axis direction- prime mode
29 gyro movement rules- axis direction- prime mode
30 gyro movement rules- axis direction- prime mode
31 gyro movement rules- axis direction- prime mode
32 gyro movement rules- axis direction- prime mode
33 gyro movement rules- axis direction- prime mode
34 gyro movement rules- axis direction- prime mode
35 gyro movement rules- axis direction- prime mode
```

36 gyro movement rules- axis direction- prime mode
37 gyro movement rules- axis direction- prime mode
38 back up mode logic for side a - no gyro
39 back up mode logic for side a - no gyro
40 back up mode logic for side a - no gyro
41 back up mode logic for side a - no gyro
42 back up mode logic for side a - no gyro
43 back up mode logic for side a - no gyro
44 back up mode logic for side a - no gyro
45 back up mode logic for side a - no gyro
46 back up mode logic for side a - no gyro
47 back up mode logic for side a - no gyro
48 back up mode logic for side a - no gyro
49 back up mode logic for side a - no gyro
50 back up mode logic for side b - no gyro
51 back up mode logic for side b - no gyro
52 back up mode logic for side b - no gyro
53 back up mode logic for side b - no gyro
54 back up mode logic for side b - no gyro
55 back up mode logic for side b - no gyro
56 back up mode logic for side b - no gyro
57 back up mode logic for side b - no gyro
58 back up mode logic for side b - no gyro
59 back up mode logic for side b - no gyro
60 back up mode logic for side b - no gyro
61 back up mode logic for side b - no gyro
62 gyro movement rules- axis direction- back-up mode
63 gyro movement rules- axis direction- back-up mode
64 gyro movement rules- axis direction- back-up mode
65 gyro movement rules- axis direction- back-up mode
66 gyro movement rules- axis direction- back-up mode
67 gyro movement rules- axis direction- back-up mode
68 gyro movement rules- axis direction- back-up mode
69 gyro movement rules- axis direction- back-up mode
70 gyro movement rules- axis direction- back-up mode
71 gyro movement rules- axis direction- back-up mode
72 gyro movement rules- axis direction- back-up mode
73 gyro movement rules- axis direction- back-up mode
74 cea-failure recovery; cea a rules
75 cea-failure recovery; cea a rules
76 cea-failure recovery; cea a rules
77 cea-failure recovery; cea a rules
78 cea-failure recovery; cea a rules
79 cea-failure recovery; cea a rules
80 cea-failure recovery; cea a rules
81 cea-failure recovery; cea b rules
82 cea-failure recovery; cea b rules
83 cea-failure recovery; cea b rules

```
 84 cea-failure recovery; cea b rules
 85 cea-failure recovery; cea b rules
 86 cea-failure recovery; cea b rules
 87 cea-failure recovery; cea b rules
 88 thruster firing failure rules
 89 thruster firing failure rules
 90 thruster firing failure rules
 91 thruster firing failure rules
 92 thruster firing failure rules
 93 thruster firing failure rules
 94 thruster firing failure rules

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
PRIMARY RULE CLUSTERING
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

MMU-FDIR : A
The primary-rules are:
38 back up mode logic for side a - no gyro
82 cea-failure recovery; cea b rules
94 thruster firing failure rules

**** DISTANCE METRIC => TOTAL ****

rules belonging to the group 38 are:
38 back up mode logic for side a - no gyro
1 improper cea behavior
2 logic for x, pitch and yaw
3 logic for x, pitch and yaw
4 logic for x, pitch and yaw
5 logic for x, pitch and yaw
6 logic for x, pitch and yaw
7 logic for x, pitch and yaw
8 logic for x, pitch and yaw
9 logic for x, pitch and yaw
10 logic for x, pitch and yaw
11 logic for x, pitch and yaw
12 logic for x, pitch and yaw
13 logic for x, pitch and yaw
14 logic for x, pitch and yaw
15  logic for y, roll and yaw
16  logic for y, roll and yaw
17  logic for y, roll and yaw
18  logic for y, roll and yaw
19  logic for y, roll and yaw
20  logic for y, roll and yaw
```

```
21  logic for y, roll and yaw
22  logic for z, roll and pitch
23  logic for z, roll and pitch
24  logic for z, roll and pitch
25  logic for z, roll and pitch
26 gyro movement rules- axis direction- prime mode
27 gyro movement rules- axis direction- prime mode
28 gyro movement rules- axis direction- prime mode
29 gyro movement rules- axis direction- prime mode
30 gyro movement rules- axis direction- prime mode
31 gyro movement rules- axis direction- prime mode
32 gyro movement rules- axis direction- prime mode
33 gyro movement rules- axis direction- prime mode
34 gyro movement rules- axis direction- prime mode
35 gyro movement rules- axis direction- prime mode
36 gyro movement rules- axis direction- prime mode
37 gyro movement rules- axis direction- prime mode
39 back up mode logic for side a - no gyro
40 back up mode logic for side a - no gyro
41 back up mode logic for side a - no gyro
42 back up mode logic for side a - no gyro
43 back up mode logic for side a - no gyro
44 back up mode logic for side a - no gyro
45 back up mode logic for side a - no gyro
46 back up mode logic for side a - no gyro
47 back up mode logic for side a - no gyro
48 back up mode logic for side a - no gyro
49 back up mode logic for side a - no gyro
50 back up mode logic for side b - no gyro
51 back up mode logic for side b - no gyro
52 back up mode logic for side b - no gyro
53 back up mode logic for side b - no gyro
54 back up mode logic for side b - no gyro
55 back up mode logic for side b - no gyro
56 back up mode logic for side b - no gyro
57 back up mode logic for side b - no gyro
58 back up mode logic for side b - no gyro
59 back up mode logic for side b - no gyro
60 back up mode logic for side b - no gyro
61 back up mode logic for side b - no gyro
62 gyro movement rules- axis direction- back-up mode
63 gyro movement rules- axis direction- back-up mode
64 gyro movement rules- axis direction- back-up mode
65 gyro movement rules- axis direction- back-up mode
66 gyro movement rules- axis direction- back-up mode
67 gyro movement rules- axis direction- back-up mode
68 gyro movement rules- axis direction- back-up mode
69 gyro movement rules- axis direction- back-up mode
```

```
70 gyro movement rules- axis direction- back-up mode
71 gyro movement rules- axis direction- back-up mode
72 gyro movement rules- axis direction- back-up mode
73 gyro movement rules- axis direction- back-up mode

rules belonging to the group 82 are:
82 cea-failure recovery; cea b rules
74 cea-failure recovery; cea a rules
75 cea-failure recovery; cea a rules
76 cea-failure recovery; cea a rules
77 cea-failure recovery; cea a rules
78 cea-failure recovery; cea a rules
79 cea-failure recovery; cea a rules
80 cea-failure recovery; cea a rules
81 cea-failure recovery; cea b rules
83 cea-failure recovery; cea b rules
84 cea-failure recovery; cea b rules
85 cea-failure recovery; cea b rules
86 cea-failure recovery; cea b rules
87 cea-failure recovery; cea b rules
90 thruster firing failure rules
91 thruster firing failure rules
92 thruster firing failure rules
93 thruster firing failure rules

rules belonging to the group 94 are:
94 thruster firing failure rules
88 thruster firing failure rules
89 thruster firing failure rules

 Average stability: 0.956044

**** DISTANCE METRIC => DATA_FLOW ****

rules belonging to the group 38 are:
38 back up mode logic for side a - no gyro
74 cea-failure recovery; cea a rules

rules belonging to the group 82 are:
82 cea-failure recovery; cea b rules
1 improper cea behavior
2 logic for x, pitch and yaw
3 logic for x, pitch and yaw
4 logic for x, pitch and yaw
5 logic for x, pitch and yaw
6 logic for x, pitch and yaw
7 logic for x, pitch and yaw
8 logic for x, pitch and yaw
```

```
9 logic for x, pitch and yaw
10 logic for x, pitch and yaw
11 logic for x, pitch and yaw
12 logic for x, pitch and yaw
13 logic for x, pitch and yaw
14 logic for x, pitch and yaw
15  logic for y, roll and yaw
16  logic for y, roll and yaw
17  logic for y, roll and yaw
18  logic for y, roll and yaw
19  logic for y, roll and yaw
20  logic for y, roll and yaw
21  logic for y, roll and yaw
22  logic for z, roll and pitch
23  logic for z, roll and pitch
24  logic for z, roll and pitch
25  logic for z, roll and pitch
26 gyro movement rules- axis direction- prime mode
27 gyro movement rules- axis direction- prime mode
28 gyro movement rules- axis direction- prime mode
29 gyro movement rules- axis direction- prime mode
30 gyro movement rules- axis direction- prime mode
31 gyro movement rules- axis direction- prime mode
32 gyro movement rules- axis direction- prime mode
33 gyro movement rules- axis direction- prime mode
34 gyro movement rules- axis direction- prime mode
35 gyro movement rules- axis direction- prime mode
36 gyro movement rules- axis direction- prime mode
37 gyro movement rules- axis direction- prime mode
39 back up mode logic for side a - no gyro
40 back up mode logic for side a - no gyro
41 back up mode logic for side a - no gyro
42 back up mode logic for side a - no gyro
43 back up mode logic for side a - no gyro
44 back up mode logic for side a - no gyro
45 back up mode logic for side a - no gyro
46 back up mode logic for side a - no gyro
47 back up mode logic for side a - no gyro
48 back up mode logic for side a - no gyro
49 back up mode logic for side a - no gyro
50 back up mode logic for side b - no gyro
51 back up mode logic for side b - no gyro
52 back up mode logic for side b - no gyro
53 back up mode logic for side b - no gyro
54 back up mode logic for side b - no gyro
55 back up mode logic for side b - no gyro
56 back up mode logic for side b - no gyro
57 back up mode logic for side b - no gyro
```

```
58 back up mode logic for side b - no gyro
59 back up mode logic for side b - no gyro
60 back up mode logic for side b - no gyro
61 back up mode logic for side b - no gyro
62 gyro movement rules- axis direction- back-up mode
63 gyro movement rules- axis direction- back-up mode
64 gyro movement rules- axis direction- back-up mode
65 gyro movement rules- axis direction- back-up mode
66 gyro movement rules- axis direction- back-up mode
67 gyro movement rules- axis direction- back-up mode
68 gyro movement rules- axis direction- back-up mode
69 gyro movement rules- axis direction- back-up mode
70 gyro movement rules- axis direction- back-up mode
71 gyro movement rules- axis direction- back-up mode
72 gyro movement rules- axis direction- back-up mode
73 gyro movement rules- axis direction- back-up mode
75 cea-failure recovery; cea a rules
76 cea-failure recovery; cea a rules
77 cea-failure recovery; cea a rules
78 cea-failure recovery; cea a rules
79 cea-failure recovery; cea a rules
80 cea-failure recovery; cea a rules
81 cea-failure recovery; cea b rules
83 cea-failure recovery; cea b rules
84 cea-failure recovery; cea b rules
85 cea-failure recovery; cea b rules
86 cea-failure recovery; cea b rules
87 cea-failure recovery; cea b rules
88 thruster firing failure rules
89 thruster firing failure rules
90 thruster firing failure rules
91 thruster firing failure rules
92 thruster firing failure rules
93 thruster firing failure rules

rules belonging to the group 94 are:
94 thruster firing failure rules

 Average stability: 0.285714

**** DISTANCE METRIC => LEFT_ONLY ****

rules belonging to the group 38 are:
38 back up mode logic for side a - no gyro
1 improper cea behavior
2 logic for x, pitch and yaw
3 logic for x, pitch and yaw
4 logic for x, pitch and yaw
```

```
5 logic for x, pitch and yaw
6 logic for x, pitch and yaw
7 logic for x, pitch and yaw
8 logic for x, pitch and yaw
9 logic for x, pitch and yaw
10 logic for x, pitch and yaw
11 logic for x, pitch and yaw
12 logic for x, pitch and yaw
13 logic for x, pitch and yaw
14 logic for x, pitch and yaw
15  logic for y, roll and yaw
16  logic for y, roll and yaw
17  logic for y, roll and yaw
18  logic for y, roll and yaw
19  logic for y, roll and yaw
20  logic for y, roll and yaw
21  logic for y, roll and yaw
22  logic for z, roll and pitch
23  logic for z, roll and pitch
24  logic for z, roll and pitch
25  logic for z, roll and pitch
26 gyro movement rules- axis direction- prime mode
27 gyro movement rules- axis direction- prime mode
28 gyro movement rules- axis direction- prime mode
29 gyro movement rules- axis direction- prime mode
30 gyro movement rules- axis direction- prime mode
31 gyro movement rules- axis direction- prime mode
32 gyro movement rules- axis direction- prime mode
33 gyro movement rules- axis direction- prime mode
34 gyro movement rules- axis direction- prime mode
35 gyro movement rules- axis direction- prime mode
36 gyro movement rules- axis direction- prime mode
37 gyro movement rules- axis direction- prime mode
39 back up mode logic for side a - no gyro
40 back up mode logic for side a - no gyro
41 back up mode logic for side a - no gyro
42 back up mode logic for side a - no gyro
43 back up mode logic for side a - no gyro
44 back up mode logic for side a - no gyro
45 back up mode logic for side a - no gyro
46 back up mode logic for side a - no gyro
47 back up mode logic for side a - no gyro
48 back up mode logic for side a - no gyro
49 back up mode logic for side a - no gyro
50 back up mode logic for side b - no gyro
51 back up mode logic for side b - no gyro
52 back up mode logic for side b - no gyro
53 back up mode logic for side b - no gyro
```

```
 54 back up mode logic for side b - no gyro
 55 back up mode logic for side b - no gyro
 56 back up mode logic for side b - no gyro
 57 back up mode logic for side b - no gyro
 58 back up mode logic for side b - no gyro
 59 back up mode logic for side b - no gyro
 60 back up mode logic for side b - no gyro
 61 back up mode logic for side b - no gyro
 62 gyro movement rules- axis direction- back-up mode
 63 gyro movement rules- axis direction- back-up mode
 64 gyro movement rules- axis direction- back-up mode
 65 gyro movement rules- axis direction- back-up mode
 66 gyro movement rules- axis direction- back-up mode
 67 gyro movement rules- axis direction- back-up mode
 68 gyro movement rules- axis direction- back-up mode
 69 gyro movement rules- axis direction- back-up mode
 70 gyro movement rules- axis direction- back-up mode
 71 gyro movement rules- axis direction- back-up mode
 72 gyro movement rules- axis direction- back-up mode
 73 gyro movement rules- axis direction- back-up mode

rules belonging to the group 82 are:
82 cea-failure recovery; cea b rules
74 cea-failure recovery; cea a rules
75 cea-failure recovery; cea a rules
76 cea-failure recovery; cea a rules
77 cea-failure recovery; cea a rules
78 cea-failure recovery; cea a rules
79 cea-failure recovery; cea a rules
80 cea-failure recovery; cea a rules
81 cea-failure recovery; cea b rules
83 cea-failure recovery; cea b rules
84 cea-failure recovery; cea b rules
85 cea-failure recovery; cea b rules
86 cea-failure recovery; cea b rules
87 cea-failure recovery; cea b rules

rules belonging to the group 94 are:
94 thruster firing failure rules
88 thruster firing failure rules
89 thruster firing failure rules
90 thruster firing failure rules
91 thruster firing failure rules
92 thruster firing failure rules
93 thruster firing failure rules

 Average stability: 0.956044
```

```
--------------------------------------------------------------------------------
AUTOMATIC RULE CLUSTERING
--------------------------------------------------------------------------------
MMU-FDIR : TOTAL : V : 3 groups

**** DISTANCE METRIC => TOTAL ****

The global stopping criteria is 3 groups:

rules belonging to the group 1 are:
1 improper cea behavior
7 logic for x, pitch and yaw
11 logic for x, pitch and yaw
19  logic for y, roll and yaw
9 logic for x, pitch and yaw
13 logic for x, pitch and yaw
20  logic for y, roll and yaw
27 gyro movement rules- axis direction- prime mode
31 gyro movement rules- axis direction- prime mode
35 gyro movement rules- axis direction- prime mode
29 gyro movement rules- axis direction- prime mode
33 gyro movement rules- axis direction- prime mode
36 gyro movement rules- axis direction- prime mode
2 logic for x, pitch and yaw
38 back up mode logic for side a - no gyro
41 back up mode logic for side a - no gyro
43 back up mode logic for side a - no gyro
5 logic for x, pitch and yaw
39 back up mode logic for side a - no gyro
6 logic for x, pitch and yaw
40 back up mode logic for side a - no gyro
10 logic for x, pitch and yaw
42 back up mode logic for side a - no gyro
22  logic for z, roll and pitch
46 back up mode logic for side a - no gyro
24  logic for z, roll and pitch
47 back up mode logic for side a - no gyro
14 logic for x, pitch and yaw
44 back up mode logic for side a - no gyro
16  logic for y, roll and yaw
45 back up mode logic for side a - no gyro
18  logic for y, roll and yaw
48 back up mode logic for side a - no gyro
49 back up mode logic for side a - no gyro
72 gyro movement rules- axis direction- back-up mode
26 gyro movement rules- axis direction- prime mode
30 gyro movement rules- axis direction- prime mode
63 gyro movement rules- axis direction- back-up mode
```

```
67 gyro movement rules- axis direction- back-up mode
64 gyro movement rules- axis direction- back-up mode
68 gyro movement rules- axis direction- back-up mode
34 gyro movement rules- axis direction- prime mode
71 gyro movement rules- axis direction- back-up mode
3 logic for x, pitch and yaw
50 back up mode logic for side b - no gyro
4 logic for x, pitch and yaw
51 back up mode logic for side b - no gyro
8 logic for x, pitch and yaw
53 back up mode logic for side b - no gyro
12 logic for x, pitch and yaw
55 back up mode logic for side b - no gyro
52 back up mode logic for side b - no gyro
54 back up mode logic for side b - no gyro
60 back up mode logic for side b - no gyro
23  logic for z, roll and pitch
58 back up mode logic for side b - no gyro
25  logic for z, roll and pitch
59 back up mode logic for side b - no gyro
15  logic for y, roll and yaw
56 back up mode logic for side b - no gyro
17  logic for y, roll and yaw
57 back up mode logic for side b - no gyro
21  logic for y, roll and yaw
61 back up mode logic for side b - no gyro
37 gyro movement rules- axis direction- prime mode
73 gyro movement rules- axis direction- back-up mode
28 gyro movement rules- axis direction- prime mode
32 gyro movement rules- axis direction- prime mode
62 gyro movement rules- axis direction- back-up mode
66 gyro movement rules- axis direction- back-up mode
65 gyro movement rules- axis direction- back-up mode
69 gyro movement rules- axis direction- back-up mode
70 gyro movement rules- axis direction- back-up mode

rules belonging to the group 2 are:
74 cea-failure recovery; cea a rules
75 cea-failure recovery; cea a rules
85 cea-failure recovery; cea b rules
78 cea-failure recovery; cea a rules
82 cea-failure recovery; cea b rules
76 cea-failure recovery; cea a rules
83 cea-failure recovery; cea b rules
79 cea-failure recovery; cea a rules
86 cea-failure recovery; cea b rules
81 cea-failure recovery; cea b rules
77 cea-failure recovery; cea a rules
```

```
84 cea-failure recovery; cea b rules
80 cea-failure recovery; cea a rules
87 cea-failure recovery; cea b rules
90 thruster firing failure rules
91 thruster firing failure rules
92 thruster firing failure rules
93 thruster firing failure rules
94 thruster firing failure rules

rules belonging to the group 3 are:
88 thruster firing failure rules
89 thruster firing failure rules

--------------------------------------------------------------------------------

MMU-FDIR : LEFT_ONLY : V : 3 groups

**** DISTANCE METRIC => LEFT_ONLY ****

The global stopping criteria is 3 groups:

rules belonging to the group 1 are:
1 improper cea behavior
7 logic for x, pitch and yaw
9 logic for x, pitch and yaw
11 logic for x, pitch and yaw
13 logic for x, pitch and yaw
19  logic for y, roll and yaw
20  logic for y, roll and yaw
27 gyro movement rules- axis direction- prime mode
31 gyro movement rules- axis direction- prime mode
35 gyro movement rules- axis direction- prime mode
29 gyro movement rules- axis direction- prime mode
33 gyro movement rules- axis direction- prime mode
36 gyro movement rules- axis direction- prime mode
2 logic for x, pitch and yaw
38 back up mode logic for side a - no gyro
41 back up mode logic for side a - no gyro
43 back up mode logic for side a - no gyro
5 logic for x, pitch and yaw
39 back up mode logic for side a - no gyro
6 logic for x, pitch and yaw
40 back up mode logic for side a - no gyro
10 logic for x, pitch and yaw
42 back up mode logic for side a - no gyro
18  logic for y, roll and yaw
48 back up mode logic for side a - no gyro
49 back up mode logic for side a - no gyro
```

72 gyro movement rules- axis direction- back-up mode
26 gyro movement rules- axis direction- prime mode
30 gyro movement rules- axis direction- prime mode
63 gyro movement rules- axis direction- back-up mode
67 gyro movement rules- axis direction- back-up mode
64 gyro movement rules- axis direction- back-up mode
68 gyro movement rules- axis direction- back-up mode
34 gyro movement rules- axis direction- prime mode
71 gyro movement rules- axis direction- back-up mode
3 logic for x, pitch and yaw
50 back up mode logic for side b - no gyro
4 logic for x, pitch and yaw
51 back up mode logic for side b - no gyro
8 logic for x, pitch and yaw
53 back up mode logic for side b - no gyro
12 logic for x, pitch and yaw
55 back up mode logic for side b - no gyro
52 back up mode logic for side b - no gyro
54 back up mode logic for side b - no gyro
60 back up mode logic for side b - no gyro
21 logic for y, roll and yaw
61 back up mode logic for side b - no gyro
37 gyro movement rules- axis direction- prime mode
73 gyro movement rules- axis direction- back-up mode
28 gyro movement rules- axis direction- prime mode
32 gyro movement rules- axis direction- prime mode
62 gyro movement rules- axis direction- back-up mode
66 gyro movement rules- axis direction- back-up mode
65 gyro movement rules- axis direction- back-up mode
69 gyro movement rules- axis direction- back-up mode
70 gyro movement rules- axis direction- back-up mode
14 logic for x, pitch and yaw
44 back up mode logic for side a - no gyro
15 logic for y, roll and yaw
56 back up mode logic for side b - no gyro
16 logic for y, roll and yaw
45 back up mode logic for side a - no gyro
17 logic for y, roll and yaw
57 back up mode logic for side b - no gyro
22 logic for z, roll and pitch
46 back up mode logic for side a - no gyro
23 logic for z, roll and pitch
58 back up mode logic for side b - no gyro
24 logic for z, roll and pitch
47 back up mode logic for side a - no gyro
25 logic for z, roll and pitch
59 back up mode logic for side b - no gyro

```
rules belonging to the group 2 are:
74 cea-failure recovery; cea a rules
81 cea-failure recovery; cea b rules
75 cea-failure recovery; cea a rules
85 cea-failure recovery; cea b rules
76 cea-failure recovery; cea a rules
78 cea-failure recovery; cea a rules
82 cea-failure recovery; cea b rules
83 cea-failure recovery; cea b rules
77 cea-failure recovery; cea a rules
84 cea-failure recovery; cea b rules
79 cea-failure recovery; cea a rules
80 cea-failure recovery; cea a rules
86 cea-failure recovery; cea b rules
87 cea-failure recovery; cea b rules
90 thruster firing failure rules
91 thruster firing failure rules
92 thruster firing failure rules
93 thruster firing failure rules
94 thruster firing failure rules

rules belonging to the group 3 are:
88 thruster firing failure rules
89 thruster firing failure rules

--------------------------------------------------------------------------------

MMU-FDIR : DATA_FLOW : V : 3 groups

**** DISTANCE METRIC => DATA_FLOW ****

The global stopping criteria is 3 groups:

rules belonging to the group 1 are:
1 improper cea behavior

rules belonging to the group 2 are:
2 logic for x, pitch and yaw
7 logic for x, pitch and yaw
9 logic for x, pitch and yaw
74 cea-failure recovery; cea a rules
75 cea-failure recovery; cea a rules
77 cea-failure recovery; cea a rules
76 cea-failure recovery; cea a rules
80 cea-failure recovery; cea a rules
78 cea-failure recovery; cea a rules
79 cea-failure recovery; cea a rules
82 cea-failure recovery; cea b rules
```

84 cea-failure recovery; cea b rules
83 cea-failure recovery; cea b rules
87 cea-failure recovery; cea b rules
85 cea-failure recovery; cea b rules
86 cea-failure recovery; cea b rules
81 cea-failure recovery; cea b rules
90 thruster firing failure rules
91 thruster firing failure rules
92 thruster firing failure rules
93 thruster firing failure rules
94 thruster firing failure rules
13 logic for x, pitch and yaw
20 logic for y, roll and yaw
29 gyro movement rules- axis direction- prime mode
33 gyro movement rules- axis direction- prime mode
36 gyro movement rules- axis direction- prime mode
11 logic for x, pitch and yaw
19 logic for y, roll and yaw
27 gyro movement rules- axis direction- prime mode
31 gyro movement rules- axis direction- prime mode
35 gyro movement rules- axis direction- prime mode
5 logic for x, pitch and yaw
6 logic for x, pitch and yaw
10 logic for x, pitch and yaw
14 logic for x, pitch and yaw
16 logic for y, roll and yaw
18 logic for y, roll and yaw
22 logic for z, roll and pitch
24 logic for z, roll and pitch
26 gyro movement rules- axis direction- prime mode
30 gyro movement rules- axis direction- prime mode
34 gyro movement rules- axis direction- prime mode
38 back up mode logic for side a - no gyro
39 back up mode logic for side a - no gyro
40 back up mode logic for side a - no gyro
41 back up mode logic for side a - no gyro
42 back up mode logic for side a - no gyro
43 back up mode logic for side a - no gyro
44 back up mode logic for side a - no gyro
45 back up mode logic for side a - no gyro
46 back up mode logic for side a - no gyro
47 back up mode logic for side a - no gyro
48 back up mode logic for side a - no gyro
49 back up mode logic for side a - no gyro
63 gyro movement rules- axis direction- back-up mode
64 gyro movement rules- axis direction- back-up mode
67 gyro movement rules- axis direction- back-up mode
68 gyro movement rules- axis direction- back-up mode

```
 71 gyro movement rules- axis direction- back-up mode
 72 gyro movement rules- axis direction- back-up mode
 3 logic for x, pitch and yaw
 4 logic for x, pitch and yaw
 8 logic for x, pitch and yaw
 12 logic for x, pitch and yaw
 15  logic for y, roll and yaw
 17  logic for y, roll and yaw
 21  logic for y, roll and yaw
 23  logic for z, roll and pitch
 25  logic for z, roll and pitch
 28 gyro movement rules- axis direction- prime mode
 32 gyro movement rules- axis direction- prime mode
 37 gyro movement rules- axis direction- prime mode
 50 back up mode logic for side b - no gyro
 51 back up mode logic for side b - no gyro
 52 back up mode logic for side b - no gyro
 53 back up mode logic for side b - no gyro
 54 back up mode logic for side b - no gyro
 55 back up mode logic for side b - no gyro
 56 back up mode logic for side b - no gyro
 57 back up mode logic for side b - no gyro
 58 back up mode logic for side b - no gyro
 59 back up mode logic for side b - no gyro
 60 back up mode logic for side b - no gyro
 61 back up mode logic for side b - no gyro
 62 gyro movement rules- axis direction- back-up mode
 65 gyro movement rules- axis direction- back-up mode
 66 gyro movement rules- axis direction- back-up mode
 69 gyro movement rules- axis direction- back-up mode
 70 gyro movement rules- axis direction- back-up mode
 73 gyro movement rules- axis direction- back-up mode

rules belonging to the group 3 are:
 88 thruster firing failure rules
 89 thruster firing failure rules
```

Report Documentation Page

NASA National Aeronautics and Space Administration

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-4372		

4. Title and Subtitle	5. Report Date
Rule Groupings: A Software Engineering Approach Towards Verification of Expert Systems	May 1991
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Mala Mehrotra	HQ91-015
	10. Work Unit No. 549-03-31-03

9. Performing Organization Name and Address	11. Contract or Grant No.
ViGYAN, Inc. 30 Research Drive, Hampton VA 23666-1325	NAS1-18585, Task 59
	13. Type of Report and Period Covered: Contractor Report

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Langley Research Center Hampton VA 23665-5225	

15. Supplementary Notes

NASA Langley Technical Monitor: Sally C. Johnson

16. Abstract

Currently, most expert system shells do not address software engineering issues for developing or maintaining expert systems. As a result, large expert systems tend to be incomprehensible, difficult to debug or modify and almost impossible to verify or validate. Partitioning rule-based systems into rule groups which reflect the underlying subdomains of the problem should enhance the comprehensibility, maintainability and reliability of expert-system software. In this paper, we elaborate our attempts to semi-automatically structure a CLIPS rule base into groups of related rules that carry the same type of information. Different distance metrics that capture relevant information from the rules for grouping are discussed. Two clustering algorithms that partition the rule base into groups of related rules are given. Two independent evaluation criteria are developed to measure the effectiveness of the grouping strategies. Results of our experiment with three sample rule bases are presented.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
rule groups verification validation knowledge-based systems clustering pattern-matching	Unclassified - Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	109	A06

NASA FORM 1626 OCT 86

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171 NASA-Langley, 1991